Module 5: Classification

Mixture Models for Classification

STAT/BIOSTAT 527, University of Washington

Emily Fox

May 30th, 2013

Overview of Classification So Far

- Supervised methods $(x_1,y_1),\dots,(x_n,y_n)$ "labeled data" as training data

$\underset{\text{input data}}{x_1}$, $\underset{\text{class label}}{y_1}$

Generative	Discriminative
LDA, QDA	logistic reg.
KDE for class.	CART
Naive Bayes	⋮
⋮	
max $P(Y,X)$	max $P(Y \mid X)$

- Objectives:

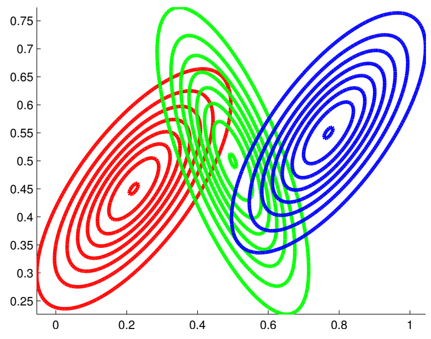
- Unsupervised methods (generative) $x_1,\dots,x_n$ "unlabeled data"

mixture models

Density as Mixture of Gaussians

- Approximate density with a mixture of Gaussians

Mixture of 3 Gaussians



$K=3$

$P = [\pi_1, \dots, \pi_K]$

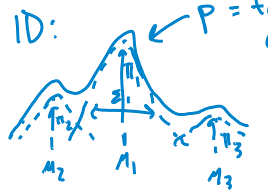
$\{\mu_k, \Sigma_k\}$

$$p(x_i \mid \pi, \mu, \Sigma) = \sum_{k=1}^{\boxed{K}} \underset{\text{mix. weights}}{\pi_k} N(x_i \mid \underbrace{\mu_k, \Sigma_k}_{\text{shape params}})$$

# of mix comp.

Gauss. kernel, just like in KDE, but not centered at obs.

In 1D:



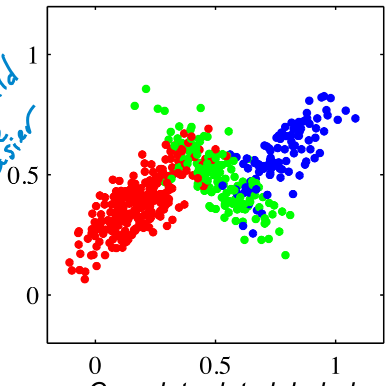
P = target density

$\sum \pi_k = 1$

Clustering our Observations

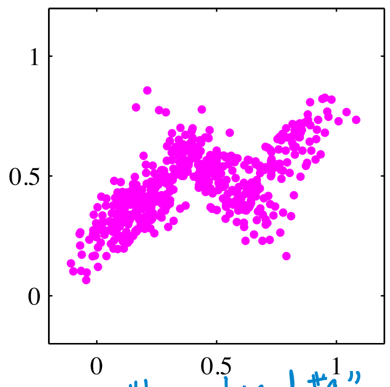
- Imagine we have an assignment of each x_i to a Gaussian

life would be easier



Complete data labeled by true cluster assignments

Our actual observations

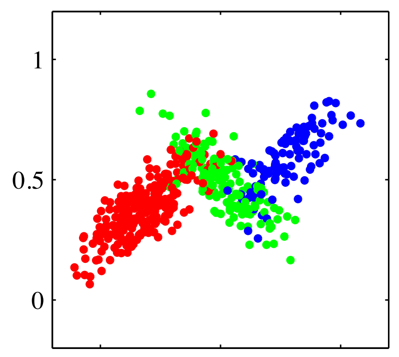


"Incomplete data"

C. Bishop, Pattern Recognition & Machine Learning

Clustering our Observations

- Imagine we have an assignment of each x_i to a Gaussian



Complete data labeled by true cluster assignments

- Introduce latent cluster indicator variable z_i

$\underline{z_i \in \{1,\dots,K\}}$ ← think of as class indicator y_i, but no training data labels

$Pr(z_i = k) = \pi_k$

- Then we have

$$p(x_i \mid \overset{k}{z_i}, \pi, \mu, \Sigma) =$$

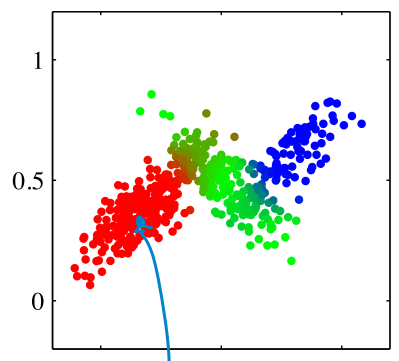
$$N(x_i \mid \mu_{\underset{k}{z_i}}, \Sigma_{\underset{k}{z_i}})$$

param. est. is easy if we have $\{z_i\}$

$\Rightarrow$ decoupled into K Gauss. est.

Clustering our Observations

- We must infer the cluster assignments from the observations



Soft assignments to clusters

- Posterior probabilities of assignments to each cluster *given* model parameters:

"responsibilities"

$$r_{ik} = p(z_i = k \mid x_i, \pi, \theta) =$$

$$= \frac{\pi_k N(x_i \mid \mu_k, \Sigma_k)}{\sum_j \pi_j N(x_i \mid \mu_j, \Sigma_j)}$$

motivates an iterative alg.

Mixture Models for Classification

- Can use mixture models as a generative classifier in the unsupervised setting

expectation maximization (EM)

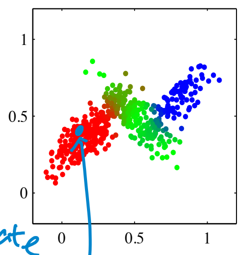
- EM algorithm = iteratively: → point est.
  - Estimate responsibilities given parameter estimates

$$\hat{r}_{ik} = \frac{\hat{\pi}_k N(x_i, \hat{\mu}_k, \hat{\Sigma}_k)}{\sum_\ell \hat{\pi}_\ell N(x_i, \hat{\mu}_\ell, \hat{\Sigma}_\ell)}$$

iterate

  - Maximize parameters given responsibilities

$\{\hat{\pi}_k, \hat{\mu}_k, \hat{\Sigma}_k\}$



- For classification, threshold the estimated responsibilities
  - E.g., $\hat{g}(x_i) = \arg\max_k \hat{r}_{ik}$

0-1 loss

x_i

r_{ik} for $k=1,2,3$

- Note: allows non-linear boundaries as in QDA

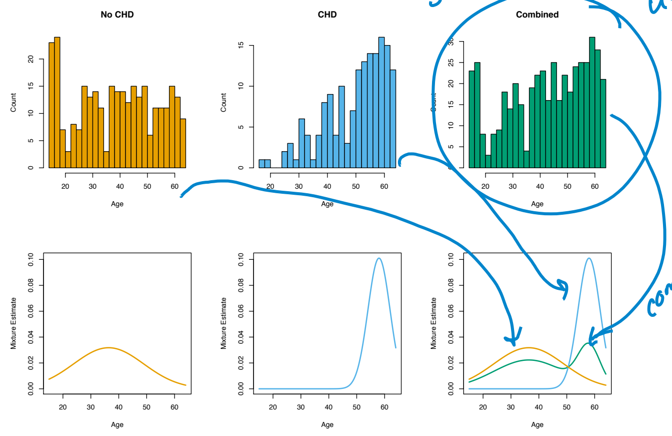
Σ_k

Example: Heart Disease Data

- Binary response = CHD (coronary heart disease)
- Predictor = ~~systolic blood pressure~~ age

using this

EM alg.



From Hastie, Tibshirani, Friedman book

What you need to know

- Discriminative vs. Generative classifiers
- LDA and QDA assume Gaussian class-conditional densities
  - Results in linear and quadratic decision boundaries, respectively
- KDE for classification
  - Challenging in areas with little data or in high dimensions
  - Estimating class-conditionals is not optimizing classification objective
- Naïve Bayes assumes factored form
  - Results in log odds that have GAM form
- Mixture models allow for unsupervised generative approach

Readings

- Hastie, Tibshirani, Friedman – 4.3, 4.4.5, 6.6.2-6.6.3, 6.8

Module 5: Classification

Online Learning Perceptron Algorithm

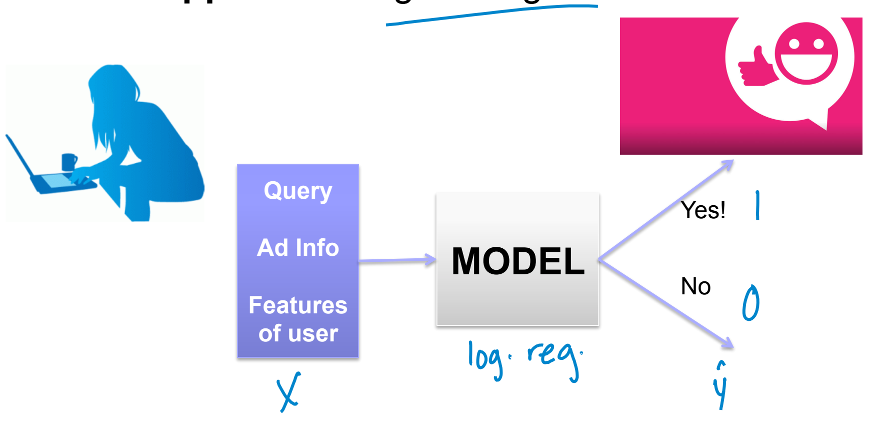
STAT/BIOSTAT 527, University of Washington

Emily Fox

May 28th, 2013

Estimating Click Probabilities

- **Goal:** Predict whether a person clicks on an ad
- **Basic approach:** Logistic regression



Challenge 1: Complexity of Computing Gradients

in terms of n, d

- What's the cost of a gradient update step for reg. LR???

$$\beta_j^{(t+1)} \leftarrow \beta_j^{(t)} + \eta\left\{-\lambda\beta_j^{(t)} + \underbrace{\sum_i x_{ij}\left(y_i - \underbrace{\hat{p}(y=1\mid x_i,\beta^{(t)})}_{O(d)}\right)}_{O(nd)}\right\}$$

For each j

$$\begin{bmatrix}\beta_1^{(t)}\\ \vdots \\ \beta_d^{(t)}\end{bmatrix}$$

instead, talked about stochastic grad. ascent ... small change after each obs.

Naively, $O(nd^2)$

but cache $\hat{p}$ (same $\forall j$) $\rightarrow O(nd)$

However, if ~~n is huge~~ (or streaming), this is very slow (infeasible) per little gradient step

Challenge 2: Data is streaming

- Assumption thus far: **Batch data**

  have data, learn model

- But, e.g., click prediction for ads is a streaming data task:
  - User enters query, and ad must be selected:
    - Observe **x**$_i$, and must predict y_i

webpage $\longrightarrow x_i$ features $\longrightarrow$ predict $\hat{y}$ (will user click)

$\longrightarrow$ which ads have high prob. of click

  - User either clicks or doesn't click on ad:
    - Label y_i is revealed afterwards
      - Google gets a reward if user clicks on ad

$\mathbb{1}(y_i \neq \hat{y})$

reward is just like 0-1 loss in class. setting

  - Weights must be updated for next time:

$\beta^{(t+1)} \leftarrow \beta^{(t)} + \Delta \leftarrow$?? what is this

Online Learning Problem

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

    X_t
  - Make a prediction:
    - Note: many models are possible, we focus on linear models

    vec. $X_{t0} = 1$?

    $\beta_0^{(t)} + \sum_j \beta_j^{(t)} X_{tj} \overset{?}{>} 0 \Rightarrow$ "click" $\|$ $\beta^{(t)} \cdot X_t \overset{?}{>} 0$
  - 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

    y_t → clicked, → not clicked
  - Update model:

    $\beta^{(t+1)} \leftarrow \beta^{(t)} + \Delta^{(t)}_{\text{something}}$

The Perceptron Algorithm [Rosenblatt '58, '62]

- Classification setting: y in {-1,+1} (note diff. from {0,1}... just a practical change)
- Linear model
  - Prediction: $\hat{y} = \text{sign}(\beta \cdot x)$
- Training:
  - Initialize weight vector: e.g. $\beta^{(0)} = 0$ (or smarter)
  - At each time step:
    - Observe covariates: X_t
    - Make prediction: $\hat{y} = \text{sign}(\beta^{(t)} \cdot x)$
    - Observe true class: $y_t \leftarrow$ true label
    - Update model:
      - If prediction is not equal to truth ... if we made mistake

If $\hat{y} = y_t$, $\beta^{(t+1)} \leftarrow \beta^{(t)}$

else, $\beta^{(t+1)} \leftarrow \beta^{(t)} + y_t X_t$

Intuition

If $\hat{y} = y_t$,

$$\beta^{(t+1)} \leftarrow \beta^{(t)}$$

else

$$\underline{\underline{\beta^{(t+1)}}} \leftarrow \beta^{(t)} + \underline{y_t x_t}$$

$$\hat{y} = \text{sign}(\beta^{(t)} \cdot x_t)$$

- Why is this a reasonable update rule?

If mistake e.g. $y_t = +1$ but $\beta^{(t)} \cdot x_t < 0$ ... wanted $\beta^{(t)} \cdot x_t > 0$

what $\max_u u \cdot x_t \rightarrow u^* = x_t$

by adding x_t to $\beta^{(t)}$, we increase $\beta^{(t)} \cdot x_t$ the most.

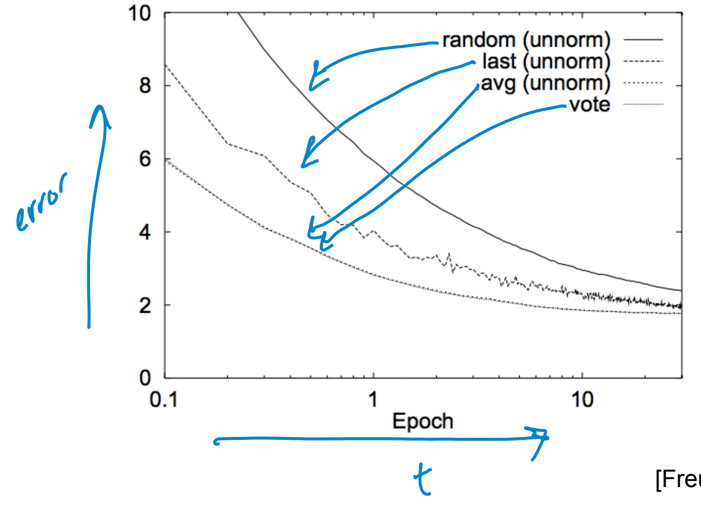
Similarly for $y_t = -1$

Which weight vector to report?

- Practical problem for all online learning methods
- Suppose you run online learning method and want to sell your learned weight vector… Which one do you sell???

- Last one? $\beta^{(T)}$? no... very noisy. Influenced by last mistake
- Random $\beta^{(\tau)} \leftarrow$ random no
- Average $\hat{\beta} = \frac{1}{T} \sum_{t=1}^{T} \beta^{(t)}$ easy to maintain
- Voting + more advanced : how long has this param been around?

Choice can make a huge difference!!



[Freund & Schapire '99]

Mistake Bounds

Why does it work?

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

Loss fcn in online setting:

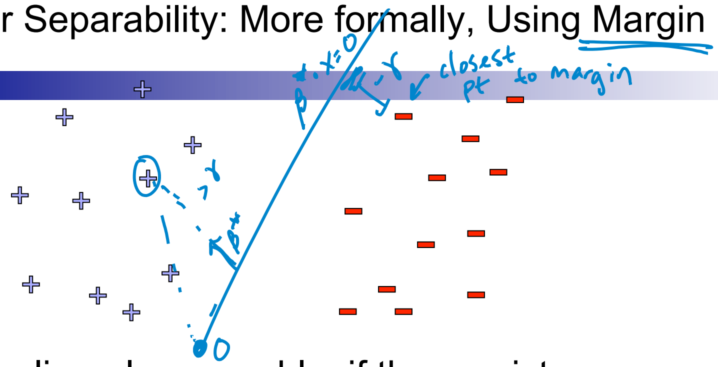
# mistakes up to time T

⇒ Google pays for its mistakes

- How many mistakes is it going to make? in its lifetime

"mistake bound"

Linear Separability: More formally, Using Margin



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

$\forall t$ if $y_t = +1 \quad \beta^* \cdot x_t > \gamma$

$y_t = -1 \quad \beta^* \cdot x_t < -\gamma \quad \Rightarrow \quad \boxed{y_t(\beta^* \cdot x_t) > \gamma}$

Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples: $(x_1, y_1), \ldots, (x_n, y_n)$
    examples need not be iid nor random
  - Each covariate vector has bounded norm:
    $\forall t \quad \|x_t\| \le R$
  - If dataset is linearly separable:
    $\exists \beta^* \quad \|\beta^*\| = 1 \quad$ s.t. $\quad y_t(\beta^* \cdot x_t) \ge \gamma \quad \leftarrow \forall t$ for some $\gamma > 0$
- Then the number of mistakes made by the online perceptron on this sequence is bounded by

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

crazy!

constant... doesn't depend on T or dim X

Perceptron Proof for Linearly Separable case

- Every time we make a mistake, we get γ closer to β*:
  - Mistake at time t: $\underline{\beta^{(t+1)} \leftarrow \beta^{(t)} + y_t x_t}$
  - Taking dot product with β*:

$$\beta^* \cdot \beta^{(t+1)} = \beta^* \cdot (\beta^{(t)} + y_t x_t)$$
$$= \beta^* \cdot \beta^{(t)} + y_t \, \beta^* \cdot x_t$$
$$\geq \beta^* \cdot \beta^{(t)} + \gamma$$

  - Thus after m mistakes:

by induction, $\beta^* \beta^{(t+1)} \geq m\gamma$

- Similarly, norm of $\beta^{(t+1)}$ doesn't grow too fast:
  - $$||\beta^{(t+1)}||^2 = ||\beta^{(t)}||^2 + \underbrace{2y_t(\beta^{(t)} \cdot x_t)}_{\substack{\text{mistake} \\ <0}} + \underbrace{||x_t||^2}_{\leq R^2}$$
  - Thus, after m mistakes:

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

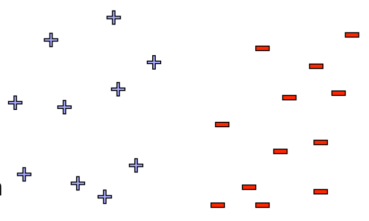
- Putting all together:

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

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

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)



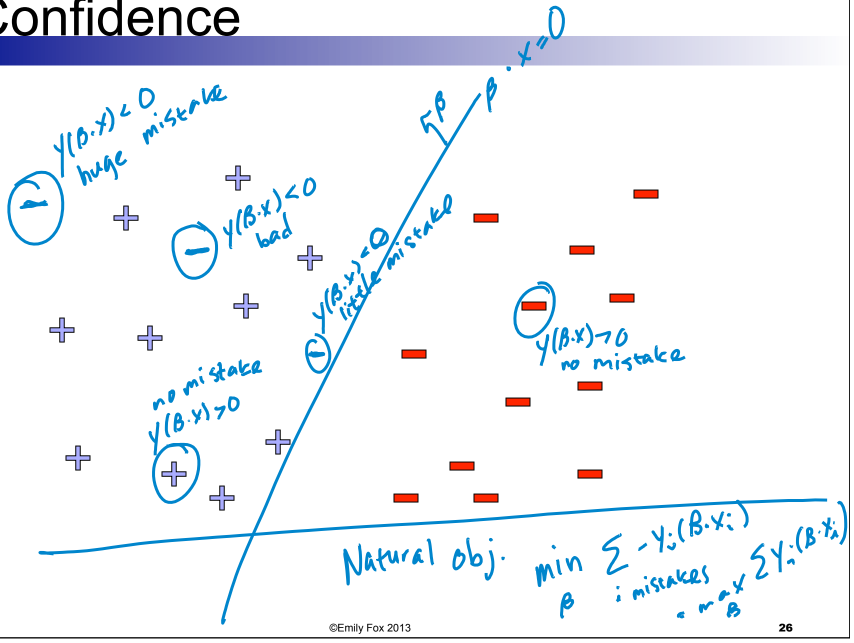
What is the Perceptron Doing???

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

$$\max_{\beta} P(Y \mid X, \beta)$$

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

Perceptron Prediction: Margin of Confidence



$\beta \cdot x = 0$
β
$y(\beta \cdot x) < 0$ huge mistake
$y(\beta \cdot x) < 0$ bad
$y(\beta \cdot x) < 0$ little mistake
$y(\beta \cdot x) > 0$ no mistake
no mistake $y(\beta \cdot x) > 0$

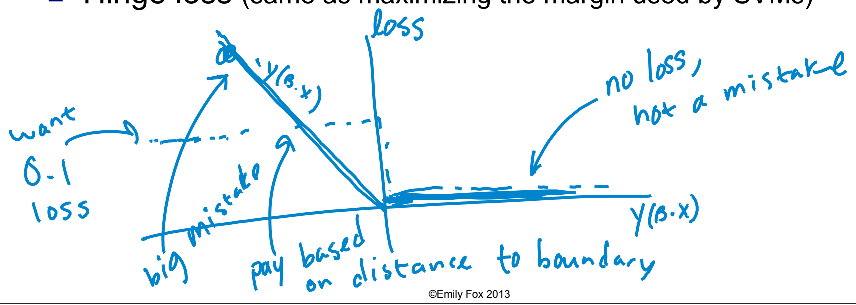
Natural obj. $\min_{\beta} \sum_{i\,\text{mistakes}} -y_i(\beta \cdot x_i) = \max_{\beta} \sum y_i(\beta \cdot x_i)$

Hinge Loss

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

$$y(\beta\cdot x) < 0 \Rightarrow \ell(\beta,x) = \begin{cases} 0 & y(\beta\cdot x) > 0 \\ -y(\beta\cdot x) & \text{else} \end{cases} \Rightarrow (-y(\beta\cdot x))_+$$

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

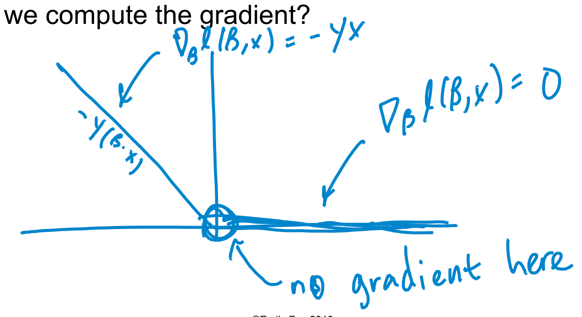


Minimizing Hinge Loss in Batch Setting

- Given a dataset: $(x_1, y_1), \ldots, (x_n, y_n)$
- Minimize average hinge loss:

$$\min_{\beta} \frac{1}{n}\sum_{i=1}^{n} \ell(\beta, x_i) \quad (-y_i(\beta\cdot x_i))_+$$

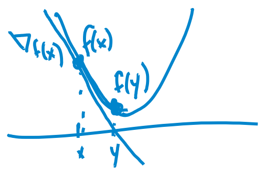
- How do we compute the gradient?



$\nabla_\beta \ell(\beta,x) = -yx$
$-y(\beta\cdot x)$
$\nabla_\beta \ell(\beta,x) = 0$
no gradient here

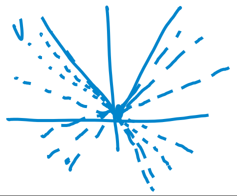
Subgradients of Convex Functions

- Gradients lower bound convex functions:



$f(y) \geq f(x) + \nabla f(x)(y-x)$

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



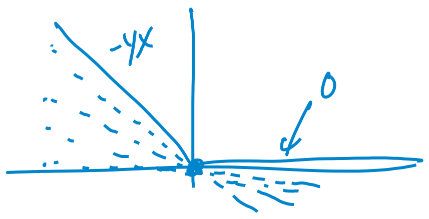
For $|\beta_j|$: $v \in [-1, 1]$

$\boxed{v \in \partial f(x) \text{ subgradient if } f(y) \geq f(x) + v(y-x)}$

©Carlos Guestrin 2013 29

Subgradient of Hinge

- Hinge loss:



- Subgradient of hinge loss:
  - If $y_t(\beta.\mathbf{x}_t) > 0$: $\partial \ell(\beta, x) = 0$
  - If $y_t(\beta.\mathbf{x}_t) < 0$: $\partial \ell(\beta, x) = -yx$
  - If $y_t(\beta.\mathbf{x}_t) = 0$: $\partial \ell(\beta, x) = [-yx, 0]$ e.g. $-yx$
  - In one line:

$$\partial \ell(\beta, x) = \underbrace{\mathbb{1}(y(\beta.x) \leq 0)}_{\text{mistake indicator}} (-yx)$$

©Emily Fox 2013 30

Subgradient Descent for Hinge Minimization

- Given data: $(x_1, y_1), \ldots, (x_n, y_n)$
- Want to minimize:

$$\frac{1}{n}\sum_{i=1}^{n}\ell(\beta, x_i) = \frac{1}{n}\sum_{i=1}^{n}(-y_i(\beta\cdot x_i))_+$$

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

$$\beta^{(t+1)} \leftarrow \beta^{(t)} - \underset{\text{step size}}{\eta} \sum_{i=1}^{n} \underbrace{\partial \ell(\beta, x_i)}_{\mathbb{1}(y_i(\beta\cdot x_i)\le 0)(-y x_i)}$$

Perceptron Revisited

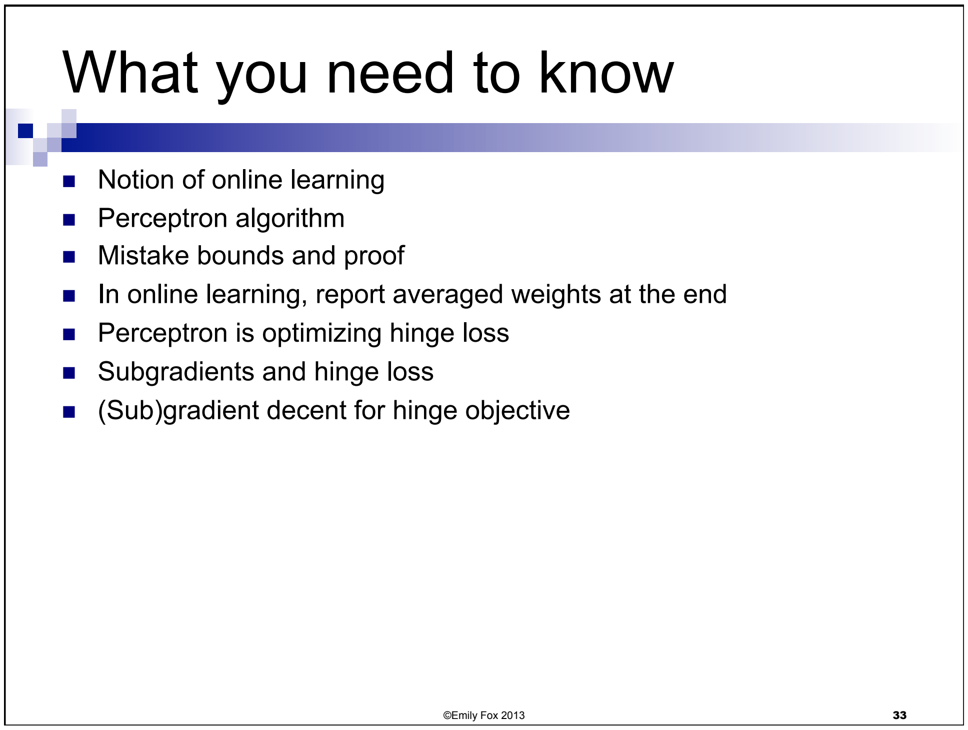
- Perceptron update:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \overset{\eta=1}{} \overbrace{\mathbb{I}\left[y_t(\beta^{(t)}\cdot x_t)\le 0\right]}^{\text{if mistake}} \overbrace{y_t x_t}^{\text{neg. subgrad.}}$$

- Batch hinge minimization update:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \underset{\text{param, hopefully } \eta > 1}{\eta}\frac{1}{n}\sum_{i=1}^{n}\left\{\overbrace{\mathbb{I}\left[y_i(\beta^{(t)}\cdot x_i)\le 0\right]}^{\text{if mistake}} \overbrace{y_i x_i}^{\text{neg. subgrad.}}\right\}$$

- Difference? Perceptron algorithm = SGD for hinge loss minimization using $\eta = 1$



What you need to know

- Notion of online learning
- Perceptron algorithm
- Mistake bounds and proof
- In online learning, report averaged weights at the end
- Perceptron is optimizing hinge loss
- Subgradients and hinge loss
- (Sub)gradient decent for hinge objective