Module 5: Classification

Mixture Models for Classification

STAT/BIOSTAT 527, University of Washington
Emily Fox
May 27th, 2014

Overview of Classification So Far

- Supervised methods $(x_1, y_1), \ldots, (x_n, y_n)$ (x: input data; y: class label)
- Objectives:

Generative	Discriminative
LDA, QDA, KDE for class, Naive Bayes	logistic regr., CART
$\max P(y, x)$	$\max P(y \mid x)$

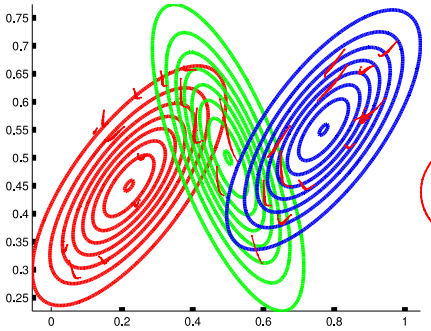
- Unsupervised methods (generative) $x_1, \ldots, x_n$

mixture models

Density as Mixture of Gaussians

- Approximate density with a mixture of Gaussians

Mixture of 3 Gaussians



K = 3

$$p(x_i \mid \pi, \mu, \Sigma) = \sum_{k=1}^{K} \pi_k N(x_i \mid \mu_k, \Sigma_k)$$

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

# of mix comp.

mix. weights

shape params

P = target density

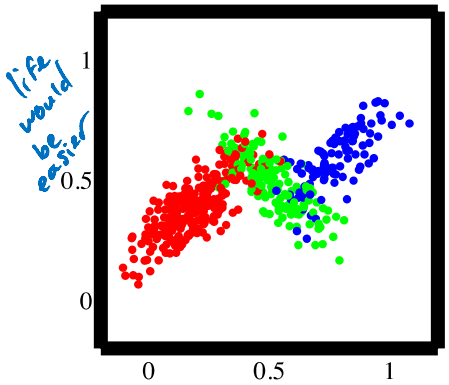
In 1D:

$\Sigma \pi_k = 1$

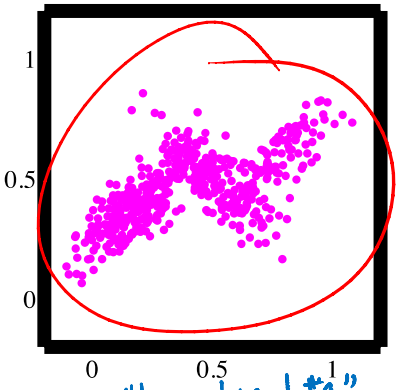
Clustering our Observations

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

Our actual observations



Complete data labeled by true cluster assignments

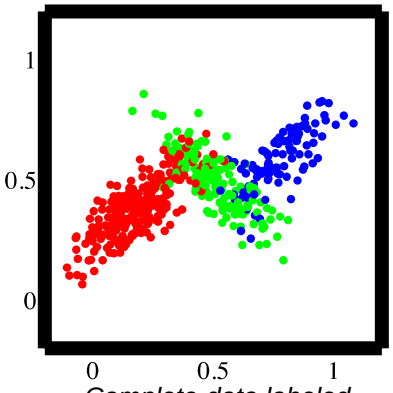


"Incomplete data"

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

$z_i \in \{1, \ldots, K\}$ ← think of as class indicator

$Pr(z_i = k) = \pi_k$

- Then we have

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

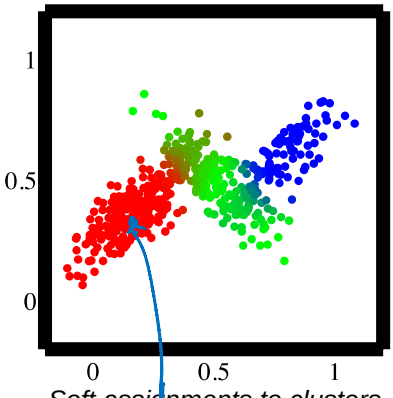
y_i, but no training data labels

param. est. is easy if we have $\{z_i\}$ ⇒ decoupled into K Gauss. est.

C. Bishop, Pattern Recognition & Machine Learning

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"

$$r_{ik} = p(z_i = k \mid x_i, \pi, \theta) = \frac{\pi_k N(x_i \mid \mu_k, \Sigma_k)}{\sum_j \pi_j N(x_i \mid \mu_j, \Sigma_j)}$$

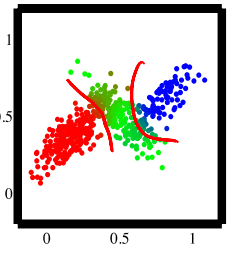
(soft cluster assignment)

motivates an iterative alg.

C. Bishop, Pattern Recognition & Machine Learning

Mixture Models for Classification

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



- EM algorithm = iteratively:
  - Estimate responsibilities given parameter estimates

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

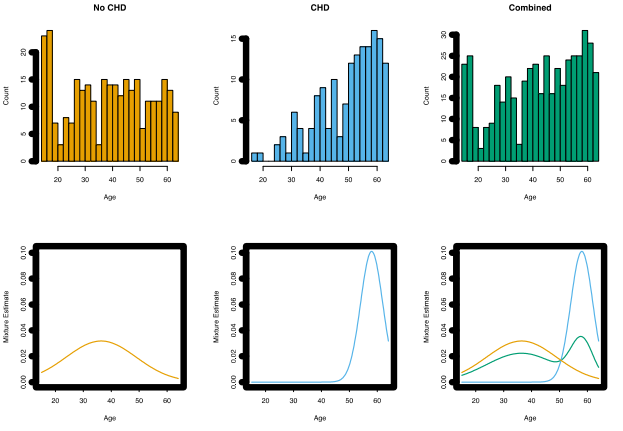
  - Maximize parameters given responsibilities

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

- Note: allows non-linear boundaries as in QDA

Example: Heart Disease Data

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



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

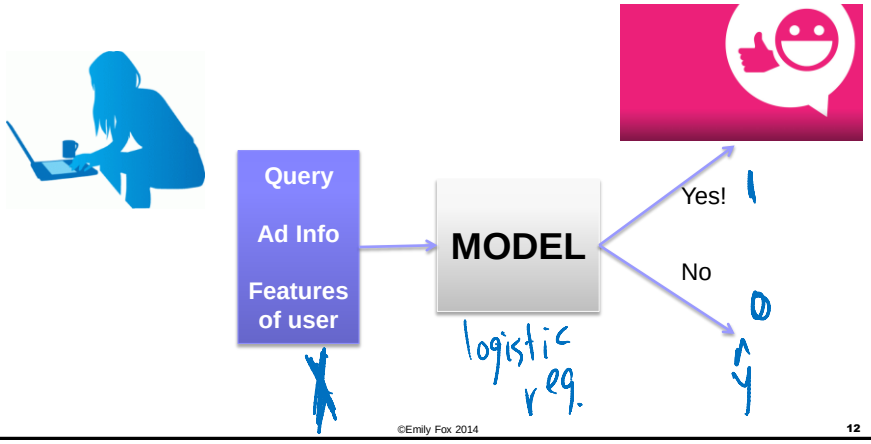
Module 5: Classification

Online Learning Perceptron Algorithm

STAT/BIOSTAT 527, University of Washington
Emily Fox
May 27th, 2014

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 LR???

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

regularized

for each j

O(d)

O(nd)

(stochastic gradient ascent)

Naively, $O(nd^2)$ but cache $\hat{p}$ (same $\forall j$) → $O(nd)$

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 dataset, run model
- But, e.g., click prediction for ads is a streaming data task:
  - User enters query, and ad must be selected:
    - Observe $\mathbf{x}_i$, and must predict y_i

webpage → x_i (features) → predicted class $\hat{y}_i$ → then observe y_i

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

goal: which ads have a high prob. of click

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

  - Weights must be updated for next time:

$\beta^{(t+1)} = \beta^{(t)} + \Delta$??, what is it?

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

$\beta_0^{(t)} + \sum_j \beta_j^{(t)} x_{t,j} > 0 \rightarrow$ "click" $\| \beta^{(t)} \cdot x_t > 0$ (vec. · vec.)

  - 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 ↕ 1 clicked, 0 not clicked

  - Update model:

$\beta^{(t+1)} = \beta^{(t)} + \Delta^{(t)}$ ← WHAT?

The Perceptron Algorithm [Rosenblatt '58, '62]

- Classification setting: y in {-1,+1} (diff. from {0,1}, practical change)
- Linear model
  - Prediction: $\hat{y} = \text{sign}(\beta \cdot x)$
- Training:
  - Initialize weight vector: $\beta^{(0)} = 0$ (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: i.e. making a mistake
      - If prediction is not equal to truth

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

else, $\beta^{(t+1)} = \beta^{(t)} + y_t x_t$

Intuition

If $\hat{y} = y_t$,

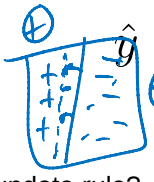
$\beta^{(t+1)} \leftarrow \beta^{(t)}$

else

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

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

- Why is this a reasonable update rule?



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

(similar logic when $y_t = -1$)

if mistake when $y_t = 1$, $\hat{y} = -1$ → $\beta^{(t)} \cdot x_t < 0$ .... want $\beta^{(t)} \cdot x_t > 0$

(what "v" maximizes $\max v \cdot x_t$ ↑ unit vector. ?)

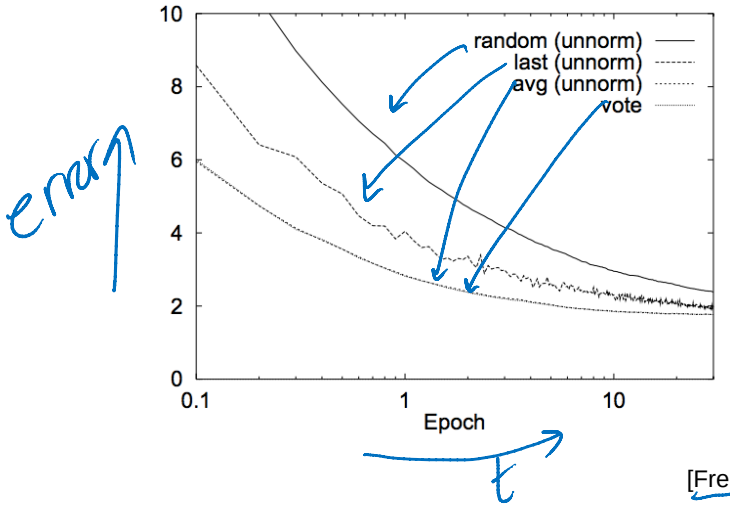
unit vector.

$\cos(\theta(v, x_t)) \propto v \cdot x_t$

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, noisy b/c influenced by last mistake.
- Random $\beta^{(\text{rand})}$
- Average $\hat{\beta} = \frac{1}{T} \sum_{t=1}^{T} \beta^{(t)}$ no (easy to maintain)
- Voting + more advanced

Choice can make a huge difference!!



[Freund & Schapire '99]

Mistake Bounds

Why does it work?

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

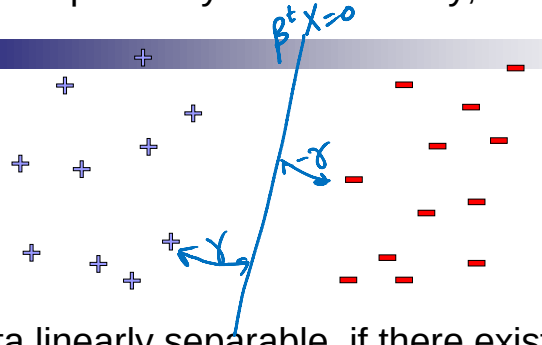
Loss fcn in online setting:
# of mistakes up to time T

→ Google "pays" for its mistakes

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

"mistake bound"

Linear Separability: More formally, Using Margin



- Data linearly separable, if there exists
  - a vector $\exists\, \beta^t \quad \|\beta^t\| = 1$
  - a margin $\gamma > 0$
- Such that

all pts. are "γ" far away or more from $\beta^t \cdot x = 0$

$\forall t$ if $y_t = 1$, $\beta^t \cdot x_t > \gamma$

$y_t = -1$, $\beta^t \cdot x_t < -\gamma \Rightarrow$ $y_t(\beta^t \cdot x_t) > \gamma$

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 or random

  - Each covariate vector has bounded norm: $\forall t \quad \|x_t\| \le R$
  - If dataset is linearly separable:

$\exists\, \beta^*$ s.t. $\|\beta^*\| = 1$ and $y_t(\beta^* x_t) > \gamma$, $\gamma > 0$. $\forall t$

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

FOREVER

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

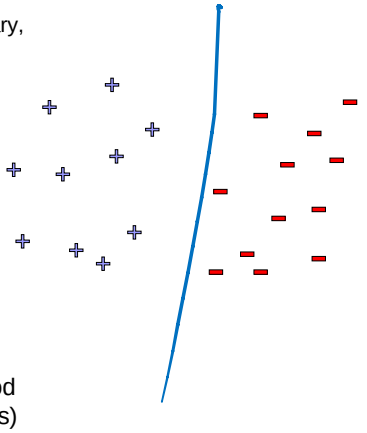
← constant... doesn't depend on T!!

Perceptron Proof for Linearly Separable case

- Every time we make a mistake, we get γ closer to β^*:
  - Mistake at time t: $\beta^{(t+1)} = \beta^{(t)} + y_t x_t$
  - Taking dot product with β^*:
  - Thus after m mistakes:
- Similarly, norm of $\beta^{(t+1)}$ doesn't grow too fast:
  - $||\beta^{(t+1)}||^2 = ||\beta^{(t)}||^2 + 2y_t(\beta^{(t)} \cdot x_t) + ||x_t||^2$
  - Thus, after m mistakes:
- Putting all together:

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

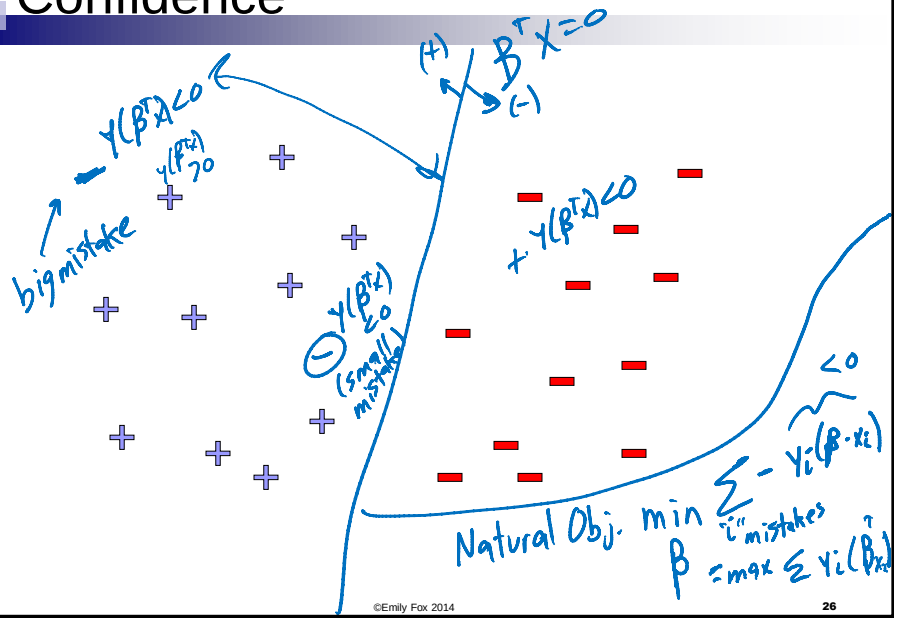
$$\max_{\beta} P(y|x)_{\beta}$$

- When we discussed the perceptron:
  - Started from description of an algorithm

- What is the perceptron optimizing???

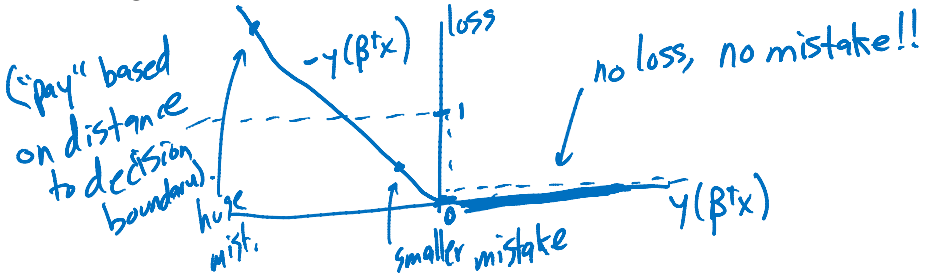
(loss fcns.)

Perceptron Prediction: Margin of Confidence



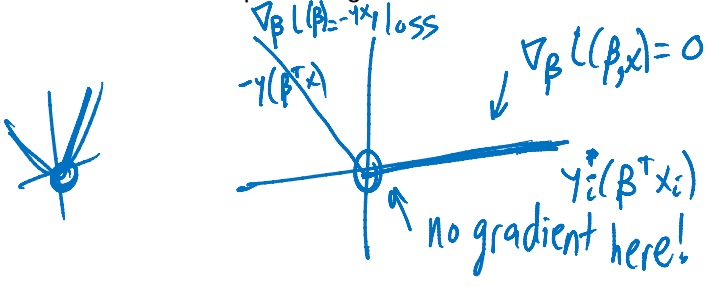
Hinge Loss

- Perceptron prediction: $sign(\beta^T x)$
- Makes a mistake when: $y(\beta^T x) < 0 \rightarrow l(\beta, x) = \begin{cases} 0 & y(\beta^T x) \geq 0 \\ -y(\beta^T x) & \text{"else"} \end{cases} \rightarrow (-y(\beta^T x))_+$
- Hinge loss (same as maximizing the margin used by SVMs)



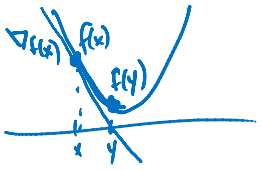
Minimizing Hinge Loss in Batch Setting

- Given a dataset: $(x_1, y_1), \ldots, (x_n, y_n)$
- Minimize average hinge loss: $\min_\beta \frac{1}{n} \sum_{i=1}^{n} l(\beta, x_i)$ ← $(-y_i(\beta^T x_i))_+$
- How do we compute the gradient?

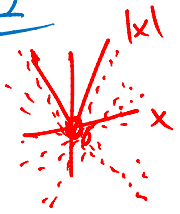


Subgradients of Convex Functions

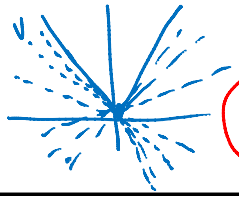
- Gradients lower bound convex functions:



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



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

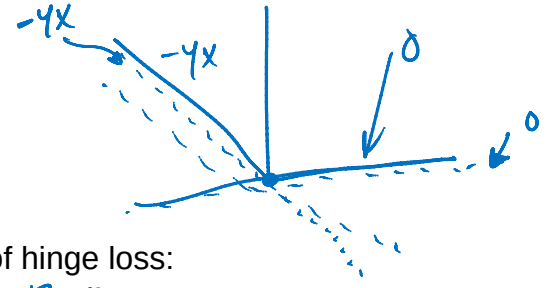


For $|\beta_j|$: $v \in [-1, 1]$

$v \in \partial f(x)$ subgradient if $f(y) \geq f(x) + v(y - x)$

Subgradient of Hinge

- Hinge loss:



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

$$\nabla_\beta \ell(\beta, x) = \mathbb{1}(y(\beta^T x) \leq 0) \cdot (-yx)$$

mistake indic.

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:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} - \eta \sum_{i=1}^{n} \nabla_\beta \ell(\beta, x_i)$$

(i.e. $-yx$)

step size

$\nabla_\beta \ell(\beta, x_i) \leftrightarrow \mathbb{I}(y_i(\beta^T x_i) \leq 0) \cdot (-yx)$

Perceptron Revisited

- Perceptron update:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \mathbb{I}\left[y_t(\beta^{(t)} \cdot x_t) \leq 0\right] y_t x_t$$

$\eta = 1$; neg. subgrad.; if mistake

- Batch hinge minimization update:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \eta \frac{1}{n}\sum_{i=1}^{n} \left\{\mathbb{I}\left[y_i(\beta^{(t)} \cdot x_i) \leq 0\right] y_i x_i\right\}$$

step; if mistake; neg. subgrad.

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

What you need to know

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