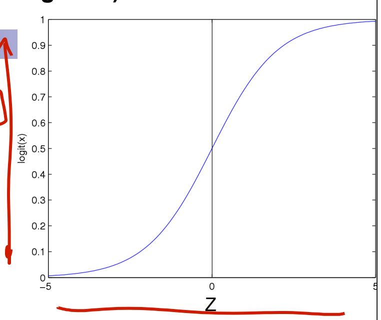
Stochastic Gradient Descent

Machine Learning – CSE546
Carlos Guestrin
University of Washington

October 14, 2014

Logistic Regression

Logistic function (or Sigmoid): $\frac{1}{1+exp(-z)}$



$\mathbb{R}$

- Learn P(Y|**X**) directly
  - Assume a particular functional form for link function $[0,1]$
  - Sigmoid applied to a linear function of the input features: $[0,1]$

$$P(Y=0|X,W) = \frac{1}{1+exp(\underbrace{w_0+\sum_i w_i X_i}_{\mathbb{R}})}$$

$$P(Y=1|X,w) = 1 - P(Y=0|X,w) = \frac{e^{w_0+\sum_i w_i X_i}}{1+e^{w_0+\sum_i w_i X_i}} \leftarrow -- \; h_i(x)$$

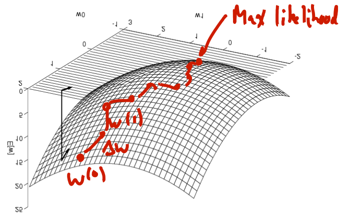
X_i

indicators: $X_i = \mathbb{1}(\text{country} = \text{USA})$

Features can be discrete or continuous!

Optimizing concave function – Gradient ascent

- Conditional likelihood for Logistic Regression is concave. Find optimum with gradient ascent



Gradient: $\nabla_{\mathbf{w}} l(\mathbf{w}) = [\frac{\partial l(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial l(\mathbf{w})}{\partial w_n}]'$

Step size, η>0

Update rule: $\Delta \mathbf{w} = \eta \nabla_{\mathbf{w}} l(\mathbf{w})$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \frac{\partial l(\mathbf{w})}{\partial w_i}$$

choice of η? from theory $\eta = \frac{\alpha}{\sqrt{t}}$ or $\eta = \frac{\alpha}{t}$ or $\eta = \alpha$ ← for HW

- Gradient ascent is simplest of optimization approaches
  - e.g., Conjugate gradient ascent can be much better

    Newton, LBFGS, ...

Gradient Ascent for LR

initialize $w^{(0)} = 0$ or something else → all converge to same place, concavity

Gradient ascent algorithm: iterate until change < ε

$$w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \sum_{j=1}^{N} [y^j - \hat{P}(Y = 1 \mid \mathbf{x}^j, \mathbf{w}^{(t)})]$$

small step

gradient direction

For i=1,…,k,

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

repeat

The Cost, The Cost!!! Think about the cost…

K features
N data points

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

same for all i
⇒ cache

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

O(NK)

for i = 0 … K

total is O(NK²)

With caching $\hat{P}$, cost is O(NK) N is really large

gradient update is very expensive for only an η step

Learning Problems as Expectations

- Minimizing loss in training data:
  - Given dataset: $X^1, X^2, \ldots X^N$
    - Sampled iid from some distribution p(**x**) on features:

      $x^j \overset{iid}{\sim} P(x)$
  - Loss function, e.g., hinge loss, logistic loss,…
  - We often minimize loss in training data:

LR. $-\ln P(y^j \mid x^j, w) + \lambda \|w\|_2^2$

$$\ell_{\mathcal{D}}(\mathbf{w}) = \frac{1}{N} \sum_{j=1}^{N} \ell(\mathbf{w}, \mathbf{x}^j)$$

approx.

- However, we should really minimize expected loss on all data:

$$\ell(\mathbf{w}) = E_{\mathbf{x}}[\ell(\mathbf{w}, \mathbf{x})] = \int p(\mathbf{x}) \ell(\mathbf{w}, \mathbf{x}) d\mathbf{x}$$

expected loss over infinite N

- So, we are approximating the integral by the average on the training data discrete: $\nabla E[f] = \nabla \sum_x f(x)\, p(x) = \sum_x \nabla f(x)\, p(x) = E_x[\nabla f(x)]$

Gradient descent in Terms of Expectations

- "True" objective function:

$$\ell(\mathbf{w}) = E_{\mathbf{x}}\left[\ell(\mathbf{w},\mathbf{x})\right] = \int p(\mathbf{x})\ell(\mathbf{w},\mathbf{x})d\mathbf{x}$$

- Taking the gradient:

$$\nabla_w \ell(w) = \nabla_w \Big(E_x [\ell(w,x)] \Big) = E_x[\nabla_w \ell(w,x)]$$

- "True" gradient descent rule:

$$w^{(t+1)} \leftarrow w^{(t)} - \eta\, E_x[\nabla_w \ell(w,x)]$$

- How do we estimate expected gradient?

SGD: Stochastic Gradient Ascent (or Descent)

- "True" gradient: $\nabla\ell(\mathbf{w}) = E_{\mathbf{x}}\left[\nabla\ell(\mathbf{w},\mathbf{x})\right]$
- Sample based approximation:

$$\nabla \ell(w) = E_x[\nabla \ell(w,x)] \approx \frac{1}{N}\sum_{j=1}^{N} \nabla_w \ell(w,x^j)$$

approx. true gradient by sample gradient over dataset

- What if we estimate gradient with just one sample???
  - Unbiased estimate of gradient $E_x[\nabla_w \ell(w,x^t)] = \nabla \ell(w)$
  - Very noisy!
  - Called stochastic gradient ascent (or descent)
    - Among many other names
  - VERY useful in practice!!!

Stochastic Gradient Ascent for Logistic Regression

- Logistic loss as a stochastic function:

$$E_{\mathbf{x}}\left[\ell(\mathbf{w},\mathbf{x})\right] = E_{\mathbf{x}}\left[\ln P(y|\mathbf{x},\mathbf{w}) - \lambda||\mathbf{w}||_2^2\right]$$

- Batch gradient ascent updates: batch approach uses all data

$O(Nk)$ per iteration

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

- Stochastic gradient ascent updates: pick a next data point $x^{(t)}, y^{(t)}$
  - Online setting: Often, in practice, mini-batches: pick, typically 10-100 data points

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \underbrace{\left\{-\lambda w_i^{(t)} + x_i^{(t)}[y^{(t)} - P(Y=1|\mathbf{x}^{(t)},\mathbf{w}^{(t)})]\right\}}_{\text{do an update as if } x^t, y^t \text{ were all the data}}$$

cost per iteration $O(k)$

Stochastic Gradient Descent: general case

- Given a stochastic function of parameters: $f(w) = E_x[f(w,x)]$
  - Want to find ~~maximum~~ minimum

$$w^* \in \arg\min_w f(w) = \arg\min_w E_x[f(w,x)]$$

- Start from $\mathbf{w}^{(0)}$ e.g. $w^{(0)} = 0$
- Repeat until convergence:
  - Get a sample data point $\mathbf{x}^t$
  - Update parameters:

$$w^{(t+1)} \leftarrow w^{(t)} - \eta_t \nabla_w f(w, x^t)$$

- Works on the online learning setting!
- Complexity of each gradient step is constant in number of examples!
- In general, step size changes with iterations

η_t decreases with iterations, from theory, typically: $\eta_t = \frac{\alpha}{t}$, $\eta_t = \alpha/\sqrt{t}$

What you should know…

- Classification: predict discrete classes rather than real values
- Logistic regression model: Linear model
  - Logistic function maps real values to [0,1]
- Optimize conditional likelihood
- Gradient computation
- Overfitting
- Regularization
- Regularized optimization
- Cost of gradient step is high, use stochastic gradient descent

Boosting

Machine Learning – CSE546
Carlos Guestrin
University of Washington

October 14, 2014

Fighting the bias-variance tradeoff

- **Simple (a.k.a. weak) learners are good**
  - e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)
  - Low variance, don't usually overfit too badly
- **Simple (a.k.a. weak) learners are bad**
  - High bias, can't solve hard learning problems
- Can we make weak learners always good???
  - **No!!!**
  - **But often yes…**

Voting (Ensemble Methods)

- Instead of learning a single (weak) classifier, learn **many weak classifiers** that are **good at different parts of the input space**
- **Output class:** (Weighted) vote of each classifier
  - Classifiers that are most "sure" will vote with more conviction
  - Classifiers will be most "sure" about a particular part of the space
  - On average, do better than single classifier!

$h_i: X \to Y \in \{-1, +1\}$

$$H(X) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

t-th classifier

Vote weight

e.g., $h_t(x) = \begin{cases} +1 & \text{if } x_i = 1 \\ -1 & \text{if } x_i = 0 \end{cases}$ if email has word 'CSE546' ⇒ not spam (−1) else ⇒ spam (+1)

- **But how do you ???**
  - force classifiers to learn about different parts of the input space?
  - weigh the votes of different classifiers?

Boosting [Schapire, 1989]

- Idea: given a weak learning alg, run it multiple times on (reweighted) training data, then let learned classifiers vote
- On each iteration t:
  - weight each training example by how incorrectly it was classified thus far
  - Learn a hypothesis – h_t
  - A strength for this hypothesis – α_t
- Final classifier: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$
- **Practically useful**
- **Theoretically interesting**

Learning from weighted data

- **Sometimes not all data points are equal**
  - Some data points are more equal than others
- **Consider a weighted dataset**
  - D(j) – weight of *j* th training example ($\mathbf{x}^j$,y^j)
  - Interpretations:
    - *j* th training example counts as D(j) examples
    - If I were to “resample” data, I would get more samples of “heavier” data points
- **Now, in all calculations, whenever used, *j* th training example counts as D(j) “examples”**

for gradient descent

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \sum_{j=1}^{N} D(j) \nabla_w \ell(w, x^j)$$

data point is weighted

Goal is to focus h_t on hard points, ones that are not classified well, by increasing their weight.

AdaBoost

- Initialize weights to uniform dist: $D_1(j) = 1/N$

uniform

- For t = 1…T
  - Train weak learner h_t on distribution D_t over the data

focus on parts with high weight

  - Choose weight α_t

$\alpha_t > 0$ magic, next slide

  - Update weights:

for each point j

old weight

new weight

$$D_{t+1}(j) = \frac{D_t(j)\exp(-\alpha_t y^j h_t(x^j))}{Z_t}$$

$y^j h_t(x_j) > 0$

$\Rightarrow$ correct class

$\exp(-\alpha_t y^j h_t(x^j)) < 1$

$\Rightarrow$ weight decreases

$y^j h_t(x_j) < 0$

$\Rightarrow$ incorrect class

$\Rightarrow \exp(-\alpha_t y^j h_t(x^j)) > 1$

$\Rightarrow$ weight increases

- Where Z_t is normalizer:

$$Z_t = \sum_{j=1}^{N} D_t(j)\exp(-\alpha_t y^j h_t(x^j))$$

renormalize so $D_t(j)$ always adds up to 1

- Output final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Picking Weight of Weak Learner

- Weigh h_t higher if it did well on training data (weighted by D_t):

$$\text{Magic: } \alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

if $\epsilon_t = 0 \Rightarrow h_t$ perfect on weighted data

$\Rightarrow h_t$ also perfect on original data

$\Rightarrow \alpha_t = \infty$

$\frac{1}{2} < \epsilon_t < 1$, h_t worse than random, but $-h_t$ is better than random

$\Rightarrow \alpha_t < 0 \Rightarrow$ flips h_t

$\epsilon_t = \frac{1}{2} \Rightarrow h_t$ is same as random guess

$\Rightarrow \alpha_t = 0 \Rightarrow$ ignore h_t

  - Where ϵ_t is the weighted training error:

$$\epsilon_t = \sum_{j=1}^{N} D_t(j)\mathbb{1}[h_t(x^j) \neq y^j]$$

Why choose α_t for hypothesis h_t this way?

[Schapire, 1989]

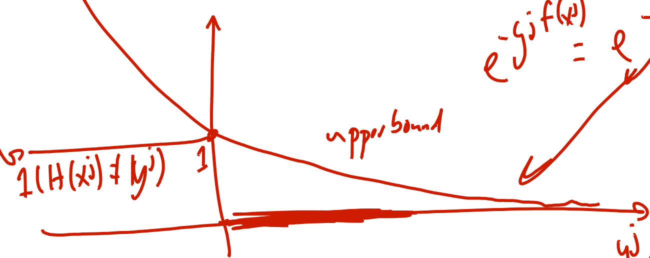
Training error of final classifier is bounded by:

training error of voted classifier

$$\frac{1}{N}\sum_{j=1}^{N}\mathbb{1}[H(x^j)\neq y^j] \leq \frac{1}{N}\sum_{j=1}^{N}\exp(-y^j f(x^j))$$

Where $f(x) = \sum_t \alpha_t h_t(x);\ H(x) = sign(f(x))$

$e^{-y^j f(x^j)} = e^{-y^j \sum_t \alpha_t h_t(x^j)}$



$> 0 \Rightarrow$ correct class

$< 0 \Rightarrow$ incorrect

$y^j f(x^j) = y^j \sum_t \alpha_t h_t(x^j)$

Why choose α_t for hypothesis h_t this way?

[Schapire, 1989]

Training error of final classifier is bounded by:

$$\boxed{Z_t = \sum_{j=1}^{N} D_t(j)\exp(-\alpha_t y^j h_t(x^j))}$$

$$\underbrace{\frac{1}{N}\sum_{j=1}^{N}\mathbb{1}[H(x^j)\neq y^j]}_{\text{training error}} \leq \underbrace{\frac{1}{N}\sum_{j=1}^{N}\exp(-y^j f(x^j))}_{\text{exponential upper bound}} = \prod_{t=1}^{T} Z_t$$

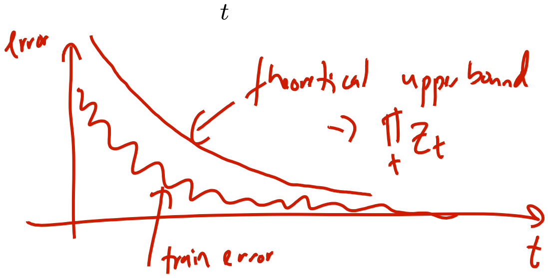
Where $f(x) = \sum_t \alpha_t h_t(x);\ H(x) = sign(f(x))$

why? see homework

magic of telescopic sum

if $Z_t < 1$

$\Rightarrow$ error bound decreases exponentially with t



Why choose α_t for hypothesis h_t this way?

[Schapire, 1989]

Training error of final classifier is bounded by:

$$\frac{1}{N}\sum_{j=1}^{N} \mathbb{1}[H(x^j) \neq y^j] \leq \frac{1}{N}\sum_{j=1}^{N} \exp(-y^j f(x^j)) = \prod_{t=1}^{T} Z_t$$

Where $f(x) = \sum_t \alpha_t h_t(x); H(x) = sign(f(x))$

If we minimize $\prod_t Z_t$, we minimize our training error

AdaBoost tightens this bound greedily, by choosing α_t and h_t on each iteration to minimize Z_t.

$$Z_t = \sum_{j=1}^{N} D_t(j) \exp(-\alpha_t y^j h_t(x^j))$$