Stochastic Gradient Descent

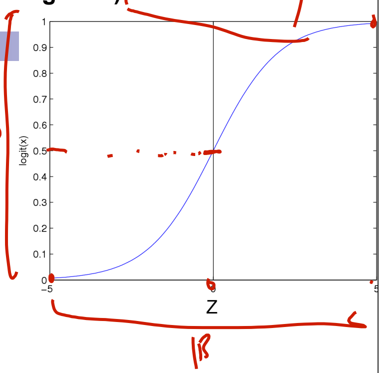
Machine Learning – CSE546

Carlos Guestrin
University of Washington

October 9, 2013

Logistic Regression

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



- 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: choice arbitrary

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

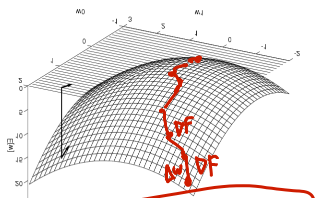
X_i

Features can be discrete or continuous!

Optimizing concave function – Gradient ascent

← alternative to coordinate ascent

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



in concave fns, grad ascent will reach OPT

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, $\eta > 0$

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

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

η choice? in theory, often $\eta = \frac{\alpha}{t}$ $\alpha > 0$ or $\eta = \frac{\alpha}{\sqrt{t}}$

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

Newtons, LBFGS,....

For your HW, choose a constant η

Gradient Ascent for LR

$w^{(0)} \leftarrow$ initialize, e.g. to 0

revisit soon

Gradient ascent algorithm: iterate until change < ε

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

For i=1,…,k,

target prediction

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

repeat

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

k features

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

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

for i = 1…k ?

total complexity is $O(Nk^2)$

But cache $\hat{P}$ before loop, then $O(Nk)$ ← if N is really large, slow… per iteration to only take 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: surrogate loss to

$$\min_w ? \; \ell_{\mathcal{D}}(\mathbf{w}) = \frac{1}{N}\sum_{j=1}^{N} \ell(\mathbf{w}, \mathbf{x}^j) \quad -\ln P(y^j \mid x^j, w) + \lambda \|w\|_2^2$$

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

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

- So, we are approximating the integral by the average on the training data

Gradient ascent in Terms of Expectations

de $\quad \nabla \alpha f = \alpha \nabla f$

- "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:

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

- "True" gradient ascent rule: (de)

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

- How do we estimate expected gradient? estimating from samples $x^j \overset{iid}{\sim} p(x)$

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: take iid samples

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

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

Stochastic Gradient Ascent for Logistic Regression

middle ground is called mini batches
↑ 10 datapoints

- 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 approach

- Batch gradient ascent updates:

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 (random) a next data point $x^{(t)}, y^{(t)}$
  - $\forall i$ Online setting:

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

O(k) per iteration

just like standard gradient, but as if dataset were one data point

Stochastic Gradient ~~A~~descent: general case

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

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

- 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 t, rate depends on properties of problem, usually $\frac{\alpha}{t} \ldots \frac{\alpha}{\sqrt{t}}$

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, 2013

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**

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

- **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!

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

t^{th} weak 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" → no spam
else spam

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

Boosting [Schapire, 1989]

ing algorithm

- Idea: given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote

$h_t(x) \rightarrow \{-1, +1\} = \hat{y}$

$y^j h_t(x^j) > 0 \Rightarrow$ correct class

$y^j h_t(x^j) < 0 \Rightarrow$ incorrect class

- On each iteration t:
  - weight each training example by how incorrectly it was classified
  - Learn a hypothesis – h_t
  - A strength for this hypothesis – α_t

decrease weight

increase weight of data point $\Rightarrow$ difficult part of space

- 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

D(j) is larger

- **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 example in gradient descent

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

data point counts D(j) times

AdaBoost

uniform weights

- Initialize weights to uniform dist: $D_1(j) = 1/N$
- For t = 1…T
  Focused on parts that have high weights
  - Train weak learner h_t on distribution D_t over the data
  - Choose weight α_t ← Magic of taking derivative & setting to $\emptyset$
  - Update weights:

    for each j

    new weight, old weight

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

    suppose $\alpha_t > 0$
    if h_t correct on j
    $y^j h_t(x^j) > 0$
    ⇒ weight decrease
    if h_t incorrect on j
    weight increases

    - Where Z_t is normalizer:

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

      so weights add 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 $\epsilon_t = 0$ ⇒ h_t perfect on weighted data
⇒ perfect on unweighted data
⇒ $\alpha_t = +\infty$

if $\frac{1}{2} > \epsilon_t > 0$
⇒ $0 < \alpha_t < +\infty$

if $\epsilon_t = 1$ ⇒ h_t perfectly wrong
⇒ $\alpha_t = -\infty$

1⇒
- Where ϵ_t is the weighted training error:

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

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

$\epsilon_t = \frac{1}{2}$ ⇒ h_t is perfectly uninformative
⇒ $\alpha_t = 0$

Why choose α_t for hypothesis h_t this way?

[Schapire, 1989]

Training error of final classifier is bounded by:

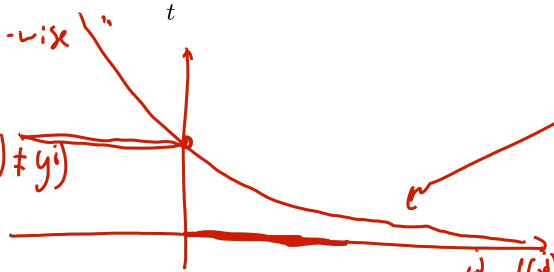
unweighted train error of classifier

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

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

element-wise

upper bound on 0/1 loss



Why choose α_t for hypothesis h_t this way?

[Schapire, 1989]

Training error of final classifier is bounded by:

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

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

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

if $0 \leq Z_t < 1$ $\forall t$: want to happen $\Rightarrow$ want to minimize Z_t

upper bound decreases exp fast

Proof: by homework, magic of telescopic sums

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

ADA Boost

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

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

Why choose α_t for hypothesis h_t this way?

[Schapire, 1989]

We can minimize this bound by choosing α_t on each iteration to minimize Z_t.

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

take derivative & set to ∅

For boolean target function, this is accomplished by [Freund & Schapire '97]:

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

You'll prove this in your homework! ☺