Case Study 1: Estimating Click Probabilities

Perceptron Algorithm
Kernels (continued)

Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington
Carlos Guestrin
January 15th, 2013

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

    $x^{(t)}$
  - Make a prediction:

    predict $y^{(t)} \approx \hat{y}$ $\quad x^{(t)} = \begin{pmatrix} x^{(t)} \\ 1 \end{pmatrix}$
    - Note: many models are possible, we focus on linear models
    - *For simplicity, use vector notation*

    $w_0 + \sum_i w_i x_i^{(t)} > 0 \, ? \Rightarrow w^{(t)} \cdot x^{(t)} > 0$

    $\sum_{i=0}^{d} w_i x_i^{(t)} \rightarrow x_0^{(t)} = 1$
  - 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 $y^{(t)} \rightarrow$ clicked or not clicked
  - Update model:

    $w^{(t+1)} \leftarrow w^{(t)} + \Delta^{(t)}$ ← … what is Δ??

    something

The Perceptron Algorithm [Rosenblatt '58, '62]

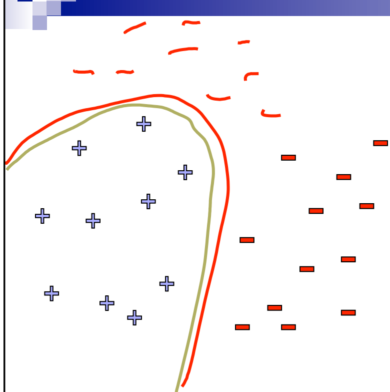
- Classification setting: y in {-1,+1}
- Linear model
  - Prediction: $\hat{y} = \text{Sign}(w \cdot x)$
- Training:
  - Initialize weight vector: $w^{(0)} = 0$
  - At each time step:
    - Observe features: $x^{(t)} \leftarrow$ user, page, ad features
    - Make prediction: $\hat{y} = \text{Sign}(w^{(t)} \cdot x^{(t)})$
    - Observe true class:
      $y^{(t)} \leftarrow$ true label
    - Update model:
      - If prediction is not equal to truth

        if make a mistake

        if $\hat{y} \neq y^{(t)}$
        $w^{(t+1)} \leftarrow w^{(t)} + y^{(t)} x^{(t)}$

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

What if the data is not linearly separable?



Use features of features of features of features….

$$\Phi(\mathbf{x}) : R^m \mapsto F$$

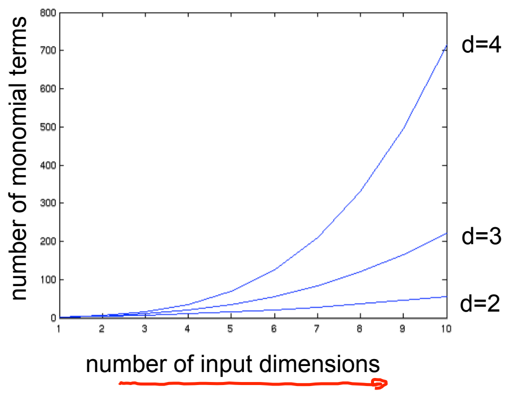
$$\phi(x) = \begin{pmatrix} x \\ x^2 \\ x^3 \\ x^4 \\ e^{-x} \\ e^{-\sin \log \cos x} \\ \vdots \end{pmatrix}$$

Feature space can get really large really quickly!

Higher order polynomials

$$\text{num. terms} = \binom{d+m-1}{d} = \frac{(d+m-1)!}{d!(m-1)!}$$

dim of ϕ



m – input features
d – degree of polynomial

$\phi(u)\cdot\phi(v) = (u \cdot v + 1)^d$

grows fast!
d = 6, m = 100
about 1.6 billion terms

Perceptron Revisited

- Given weight vector $w^{(t)}$, predict point $\mathbf{x}$ by:

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

- Mistake at time t: $w^{(t+1)} = w^{(t)} + y^{(t)} x^{(t)}$

- Thus, write weight vector in terms of mistaken data points only:
  - Let $M^{(t)}$ be time steps up to t when mistakes were made:

  $w^{(t)} = \sum_{i \in M(t)} y^{(i)} x^{(i)}$

- Prediction rule now:

  $\text{sign}(w^{(t)} \cdot x) = \text{sign}\left(\left(\sum_{i \in M(t)} y^{(i)} x^{(i)}\right) \cdot x\right) = \text{sign}\left(\sum_{i \in M(t)} y^{(i)} x^{(i)} \cdot x\right)$

- When using high dimensional features: ← list of mistakes ever made

  $\text{sign}\left(\sum_{i \in M(t)} y^{(i)} \phi(x^{(i)}) \cdot \phi(x)\right)$

  dot product between x and mistake i

Dot-product of polynomials

$u = (u_1, u_2)$
$v = (v_1, v_2)$

$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) =$ polynomials of degree exactly d

$d=1 \quad \phi(u)\cdot\phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = u.v$

$d=2 \quad \phi(u)\cdot\phi(v) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} \cdot \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} = u_1^2 v_1^2 + 2 u_1 u_2 v_1 v_2 + u_2^2 v_2^2 = (u_1 v_1 + u_2 v_2)^2 = (u.v)^2$

proof by one step of induction

if $\phi(\cdot)$ is poly of degree exactly d,

$\phi(u)\cdot\phi(v) = (u.v)^d$

compute in time of basically u.v

Finally the Kernel Trick!!! (Kernelized Perceptron

- Every time you make a mistake, remember $(x^{(t)}, y^{(t)})$

↳ Keep indices $M(t)$ of mistakes up to t & $x^{(i)}, y^{(i)}$ for these mistakes

- Kernelized Perceptron prediction for **x**:

$$\text{sign}(\mathbf{w}^{(t)} \cdot \phi(\mathbf{x})) = \sum_{i \in M^{(t)}} y^{(i)} \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x})$$

predict at any x

$$= \sum_{i \in M^{(t)}} y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x})$$

$= \phi(x^{(i)}) \cdot \phi(x)$

Polynomial kernels

- All monomials of degree d in O(d) operations:

$$\underline{\Phi(\mathbf{u})\cdot\Phi(\mathbf{v})} = (\underline{\mathbf{u}\cdot\mathbf{v}})^d = \text{polynomials of degree exactly d}$$

- How about all monomials of degree up to d?
  - Solution 0: $\phi(u)\cdot\phi(v) = \sum_{i=0}^{d} \binom{d}{i} (u\cdot v)^i$ ← about $O(d^2)$
  - Better solution: $d=2$

$(u\cdot v)^1 + (u\cdot v)^2 + (u\cdot v)^0 + (v\cdot u)^1 = (u\cdot v+1)^2$

Kernels for all monomials of degree up to and including d

$K(u,v) = \phi(u)\cdot\phi(v) = (u\cdot v+1)^d$ ← about $O(d)$

Common kernels

- Polynomials of degree exactly d ✓

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^d$$

- Polynomials of degree up to d ✓

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v}+1)^d$$

- Gaussian (squared exponential) kernel ✓ radial basis functions ...

$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||^2}{2\sigma^2}\right)$$

infinite dimensional feature spaces

- Sigmoid

$$K(\mathbf{u},\mathbf{v}) = \tanh(\eta\mathbf{u}\cdot\mathbf{v}+\nu)$$

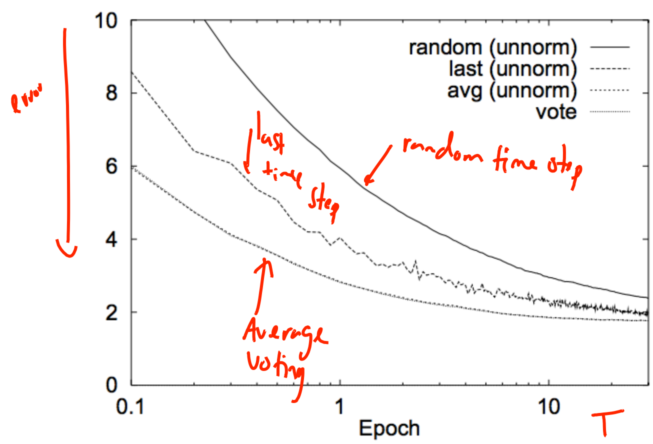
? on strings

on graphs ...

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

- Suppose you run online learning method and want to sell your learned weight vector… Which one do you sell???
- Last one? $w^{(T)}$ ← can be noisy, highly affected by last mistake
- Random time step?
- average!! ☺ $\hat{w} = \frac{1}{T}\sum_{t=0}^{T} w^{(t)}$ (easy to keep track of sum) ← very important
- voting, see Freund, Shapire from readings

Choice can make a huge difference!!



[Freund & Schapire '99]

What you need to know

- Notion of online learning
- Perceptron algorithm
- Mistake bounds and proofs
- The kernel trick
- Kernelized Perceptron
- Derive polynomial kernel
- Common kernels
- In online learning, report averaged weights at the end

in online settings, often user preferences change over time.
can address this by averaging biased towards recent wts

Case Study 1: Estimating Click Probabilities

Stochastic Gradient Descent

Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington
Carlos Guestrin
January 15th, 2013

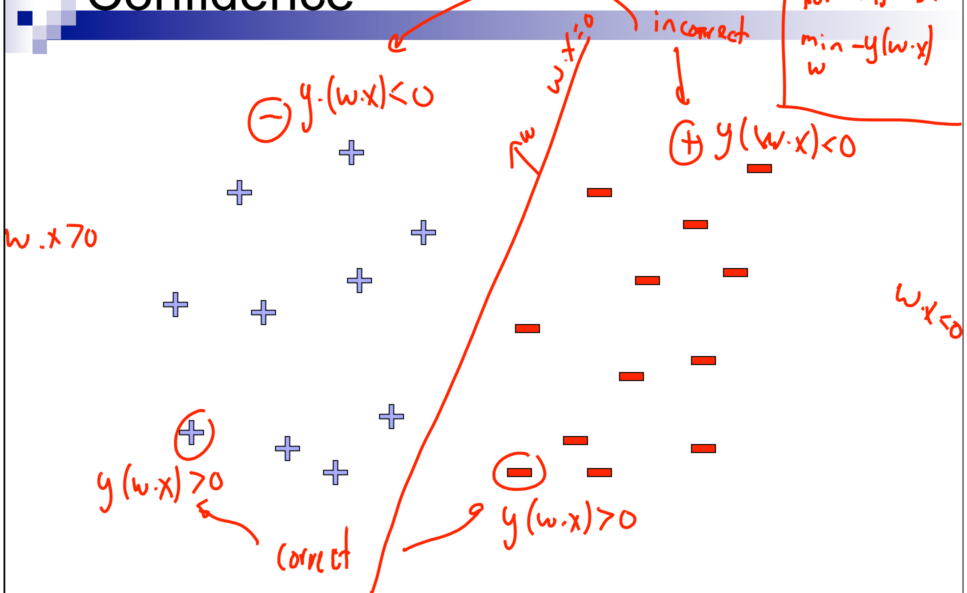
What is the Perceptron Doing???

- When we discussed logistic regression:
  - Started from maximizing conditional log-likelihood
- When we discussed the Perceptron:
  - Started from description of an algorithm
- What is the Perceptron optimizing????

Perceptron Prediction: Margin of Confidence

natural object for mistakes:

$\min_w -y(w\cdot x)$

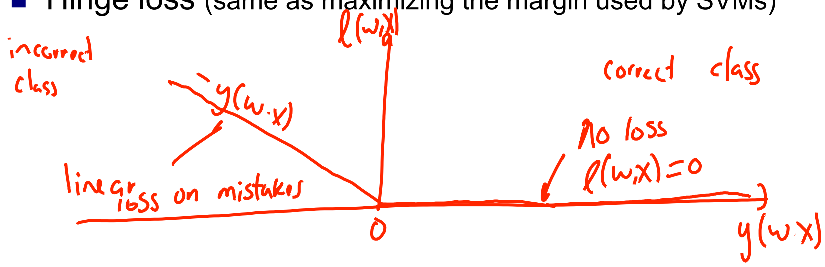


Hinge Loss

- Perceptron prediction: $\text{sign}(w \cdot x)$
- Makes a mistake when:
  $y(w \cdot x) < 0$

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

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

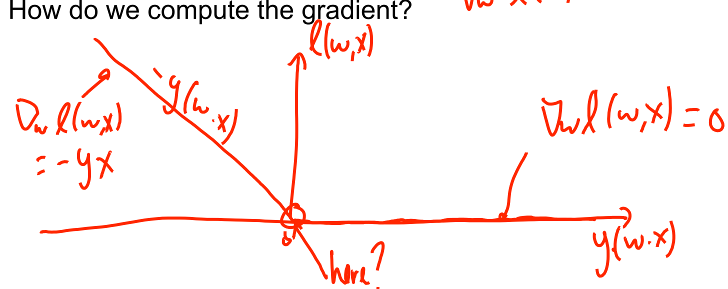


Minimizing hinge loss in Batch Setting

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

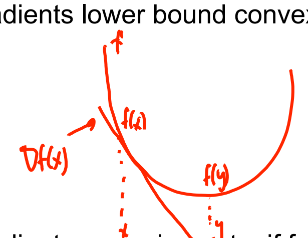
$$\min_w \frac{1}{N} \sum_{i=1}^{N} \underbrace{(-y^{(i)}(w \cdot x^{(i)}))_+}_{\ell(w,x)}$$

- How do we compute the gradient? $\nabla_w \ell(w,x)$?



Subgradients of Convex Functions

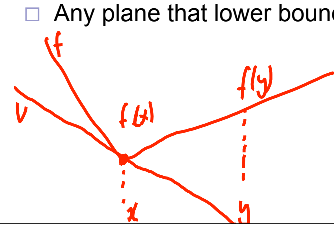
- Gradients lower bound convex functions:



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

- Gradients are unique at **x** if function differentiable at **x**

- Subgradients: Generalize gradients to non-differentiable points:
  - Any plane that lower bounds function:

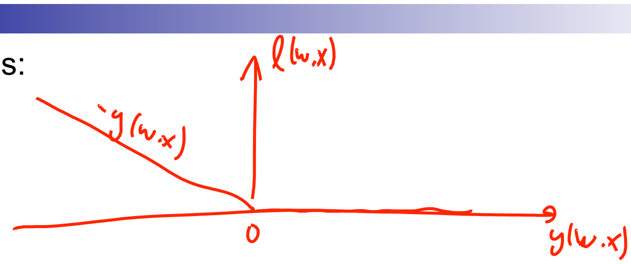


$v \in \partial f(x)$ subgradient if

$f(y) \geq f(x) + v\cdot(y-x)$

Subgradient of Hinge

- Hinge loss:



- Subgradient of hinge loss:
  - If $y^{(t)}(w.\mathbf{x}^{(t)}) > 0$: $\partial \ell(w, x^{(t)}) = 0$
  - If $y^{(t)}(w.\mathbf{x}^{(t)}) < 0$: $\partial \ell(w, x^{(t)}) = -yx$
  - If $y^{(t)}(w.\mathbf{x}^{(t)}) = 0$: $\partial \ell(w, x^{(t)}) = [-yx, 0]$, e.g., choose $-yx$
  - In one line:

$\partial \ell(w, x^{(t)}) = \mathbb{1}(y^{(t)}(w.x^{(t)}) \leq 0)\,[-yx]$

↑ think of this as gradient of hinge loss

Announcements

- No recitation this week
- Comments on readings:
  - Material in readings are superset of what you need
  - Read foundations, e.g., from Kevin Murphy's book, before class
  - Fill in details after class
- Homework 1 out today
  - Start early, start early, start early, start early, start early, start early, start early, start early, start early, start early, start early, start early, start early…
  - Warm-up part of programming due on 1/22
  - Full homework due on 1/29, beginning of class

Subgradient Descent for Hinge Minimization

- Given data: $(x^{(1)}, y^{(1)}) \ldots (x^{(N)}, y^{(N)})$
- Want to minimize:

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

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

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

Perceptron Revisited

- Perceptron update:

when mistakes were made

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

Step size = 1

- Batch hinge minimization update:

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

step size

average over points

- Difference?

Learning Problems as Expectations

- Minimizing loss in training data:
  - Given dataset: $x^{(1)} \ldots x^{(N)}$
    - Sampled iid from some distribution p(**x**) on features: $x^{(i)} \overset{iid}{\sim} p(x)$
  - Loss function, e.g., hinge loss, logistic loss,…
  - We often minimize loss in training data:

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

monte carlo integration

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

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

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

Gradient descent in Terms of Expectations

- "True" objective function:

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

- 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 descent rule:

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

- How do we estimate expected gradient? estimate expected gradient to solve true loss

©Carlos Guestrin 2013 25

SGD: Stochastic Gradient Descent (or Ascent)

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

- Sample based approximation: take iid samples $x^{(i)}$

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

the bigger N, the closer $\hat{\nabla}\ell$ is to $\nabla \ell$

- What if we estimate gradient with just one sample??? $x^{(t)}$ iid $\sim p(x)$
  - Unbiased estimate of gradient $\nabla \ell(w) \approx \hat{\nabla} \ell(w) = \nabla \ell(w, x^{(t)})$
    $E[\hat{\nabla} \ell(w)] = \nabla \ell(w)$
  - Very noisy!
  - Called stochastic gradient descent
    - Among many other names
  - VERY useful in practice!!!

©Carlos Guestrin 2013 26

Perceptron & Stochastic Gradient descent

complexity in terms of N : O(1)

- Perceptron update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \underbrace{\mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}}_{\text{1 sample estimate of } E[\nabla \ell(w)]}$$

Stochastic Gradient descent!! with $\eta = 1$

O(N)

- Batch hinge minimization update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \underbrace{\eta \frac{1}{N} \sum_{i=1}^{N} \left\{ \mathbb{1}\left[y^{(i)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(i)}) \leq 0\right] y^{(i)}\mathbf{x}^{(i)} \right\}}_{\text{N sample estimate of } E[\nabla \ell(w)]}$$

Stochastic Gradient Descent: general case

after sufficiently large t, running avg of loss approx. true loss, use for stopping.

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

    $w^* \leftarrow \arg\min_w E_x[f(w,x)]$
- Start from $\mathbf{w}^{(0)}$; e.g., $w^{(0)} = 0$
- Repeat until convergence:
  - Get a sample data point $\mathbf{x}^{(t)}$

    → sell an ad & get feedback
  - Update parameters:

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

Stopping criteria very hard... → theory bounds # iterations in terms of uncomputable constants → in practice, validation set

- Works on the online learning setting!
- Complexity of gradient step computation is constant in number of examples!
- In general, step size changes with iterations, e.g., $\eta_t = \frac{K}{t}$ for $K > 0$

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:

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

average over N datapoints

- Stochastic gradient ascent updates:
  - Online setting:

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

1 point at a time