Case Study 1: Estimating Click Probabilities

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
  - Make a prediction:
    - Note: many models are possible, we focus on linear models
    - For simplicity, use vector notation
  - 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
  - Update model:
    - $w(t+1) = w(t) + \Delta$ (something)
The Perceptron Algorithm

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:
  - Make prediction:
  - Observe true class:
  - Update model:
    - If prediction is not equal to truth

What if the data is not linearly separable?

Use features of features of features of features:

\[ \Phi(x) : \mathbb{R}^m \mapsto F \]

Feature space can get really large really quickly!
Higher order polynomials

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

- \( m \) – input features
- \( d \) – degree of polynomial

- grows fast!
- \( d = 6, m = 100 \)
- about 1.6 billion terms

Perceptron Revisited

- Given weight vector \( w^{(i)} \), predict point \( x \) by:
  \[ y = \text{Sign} (w^{(i)} \cdot x) \]

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

- Thus, write weight vector in terms of mistaken data points only:
  - Let \( M^{(i)} \) be time steps up to \( t \) when mistakes were made:
    \[ w^{(i)} = \sum_{i \in M^{(i)}} y^{(i)} x^{(i)} \]

- Prediction rule now:
  \[ \text{Sign} (w^{(i)} \cdot x) = \text{Sign} \left( \sum_{i \in M^{(i)}} y^{(i)} x^{(i)} \cdot x \right) = \text{Sign} \left( \sum_{i \in M^{(i)}} y^{(i)} x^{(i)} \cdot x \right) \]

- When using high dimensional features:
  \[ \text{Sign} \left( \sum_{i \in M^{(i)}} y^{(i)} \phi(x^{(i)}) \cdot \phi(x^{(i)}) \right) \]
  \[ \text{dot product between } x \text{ and mistakes} \]
Dot-product of polynomials

\[ \Phi(u) \cdot \Phi(v) = \text{polynomials of degree exactly } d \]

\[ \Phi(u) \cdot \Phi(v) = (u_1, u_2, \ldots, u_d) \cdot (v_1, v_2, \ldots, v_d) = u_1v_1 + \cdots + u_dv_d = u \cdot v \]

**Proof by one step of induction**

If \( \Phi() \) is poly of degree exactly \( d \),

\[ \Phi(u) \cdot \Phi(v) = (u \cdot v)^d \]

\[ \text{compute in the } d \text{th basic poly } u \cdot v \]

---

Finally the Kernel Trick!!!

**Kernelized Perceptron**

- Every time you make a mistake, remember \((x^{(t)}, y^{(t)})\)
  
  \[ \text{Keep indices } M(t) \text{ of mistakes up to } t \]
  
  \[ \text{predict } x^{(t)} \text{ for these mistakes} \]

- Kernelized Perceptron prediction for \(x\):

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

\[ \text{predict at any } x = \sum_{i \in M(t)} y^{(i)} k(x^{(i)}, x) \cdot \phi(x) \]
Polynomial kernels

- All monomials of degree d in $O(d)$ operations:
  $$\Phi(u) \cdot \Phi(v) = (u \cdot v)^d$$
  polynomials of degree exactly d

- How about all monomials of degree up to d?
  
  □ Solution 0: $$\sum_{i=0}^{d} \binom{d}{i} (u \cdot v)^i$$
  
  □ Better solution:
  $$\sum_{k=1}^{d} (u \cdot v)^k = (u \cdot v + 1)^d$$

Common kernels

- Polynomials of degree exactly d
  $$K(u, v) = (u \cdot v)^d$$

- Polynomials of degree up to d
  $$K(u, v) = (u \cdot v + 1)^d$$

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

- Sigmoid
  $$K(u, v) = \tanh(\eta u \cdot v + \nu)$$
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?
  - Random time step?
  - Average!! \( \bar{w} = \frac{1}{T+20} \sum_{t=0}^{T} w(t) \) (easy to keep track of)
  - Voting, see Freund, Schapire 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

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????
Hinge Loss

- Perceptron prediction: \( \text{Sign}(w \cdot x) \)
- Makes a mistake when: \( y(w \cdot x) < 0 \)
- Hinge loss (same as maximizing the margin used by SVMs)

\[
\ell(w, x) = \begin{cases} 
0 & \text{if } y(w \cdot x) > 0 \\
-y(w \cdot x), & \text{otherwise}
\end{cases}
\]

\[\ell(w, x) = -y \cdot (w \cdot x) \]

Minimizing hinge loss in Batch Setting

- Given a dataset: \( (x^{(1)}, y^{(1)}) \ldots (x^{(N)}, y^{(N)}) \)
- Minimize average hinge loss:
  \[
  \min_w \frac{1}{N} \sum_{i=1}^{N} \ell(y^{(i)}(w \cdot x^{(i)}))
  \]
- How do we compute the gradient?

\[\nabla \ell(w, x) = -y x \]
Subgradients of Convex Functions

- Gradients lower bound convex functions:
  \[ f(y) \geq f(x) + \nabla f(x) \cdot (y-x) \]

- Gradients are unique at \( x \) if function differentiable at \( x \)

- Subgradients: Generalize gradients to non-differentiable points:
  - Any plane that lower bounds function:
    \[ v \in \partial f(x) \text{ is a subgradient} \]
    \[ f(y) \geq f(x) + v \cdot (y-x) \]

Subgradient of Hinge

- Hinge loss:

- Subgradient of hinge loss:
  - If \( y^{(i)} (w \cdot x^{(i)}) > 0 \):
    \[ \nabla l(w, x^{(i)}) = 0 \]
  - If \( y^{(i)} (w \cdot x^{(i)}) < 0 \):
    \[ \nabla l(w, x^{(i)}) = -y x \]
  - If \( y^{(i)} (w \cdot x^{(i)}) = 0 \):
    \[ \nabla l(w, x^{(i)}) = [-y x, 0] \text{, e.g., choose } -y x \]
  - In one line:
    \[ \nabla l(w, x^{(i)}) = \mathbb{1}(y^{(i)}(w \cdot x^{(i)}) \leq 0) [-y x] \]

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 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...
  - 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: \((x_i^0, y_i^0), \ldots, (x_i^N, y_i^N)\)

- Want to minimize:
  \[
  \frac{1}{N} \sum_{i=1}^{N} \ell(w, x_i^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} (1 - y_i^{(i)})(w \cdot x_i^{(i)})
  \]

- Subgradient descent works the same as gradient descent:
  - But if there are multiple subgradients at a point, just pick (any) one:
  \[
  w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i^{(i)}(w \cdot x_i^{(i)}) \leq 0) \left[ -y_i^{(i)} x_i^{(i)} \right]
  \]
Perceptron Revisited

- Perceptron update:
  \[ \mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta \left( \mathbb{1} \left[ y^{(t)} \left( \mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)} \right) \leq 0 \right] y^{(t)} \mathbf{x}^{(t)} \right) \]

- Batch hinge minimization update:
  \[ \mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta \frac{1}{N} \sum_{i=1}^{N} \left\{ \mathbb{1} \left[ y^{(i)} \left( \mathbf{w}^{(t)} \cdot \mathbf{x}^{(i)} \right) \leq 0 \right] y^{(i)} \mathbf{x}^{(i)} \right\} \]

- Difference?

Learning Problems as Expectations

- Minimizing loss in training data:
  - Given dataset:
    - Sampled iid from some distribution \( p(\mathbf{x}) \) on features:
  - Loss function, e.g., hinge loss, logistic loss, etc.
  - We often minimize loss in training data:
    \[ \min_{\mathbf{w}} \ell_{\mathcal{D}}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\mathbf{w}, \mathbf{x}^{(i)}) \]

- However, we should really minimize expected loss on all data:
  \[ \min_{\mathbf{w}} \ell(\mathbf{w}) = \mathbb{E}_{\mathbf{x}} [\ell(\mathbf{w}, \mathbf{x})] = \int p(\mathbf{x}) \ell(\mathbf{w}, \mathbf{x}) d\mathbf{x} \]

- So, we are approximating the integral by the average on the training data.
Gradient descent in Terms of Expectations

- "True" objective function:
  \[
  \min w \{ \ell(w) = E_x [\ell(w, x)] = \int p(x) \ell(w, x) dx \}
  \]

- Taking the gradient:
  \[
  \nabla w \ell(w) = \nabla w \left[ E_x [\ell(w, x)] \right] = \mathbb{E}_x [\nabla w \ell(w, x)]
  \]

- "True" gradient descent rule:
  \[
  w(t+1) \leftarrow w(t) - \eta \mathbb{E}_x [\nabla w \ell(w, x)]
  \]

- How do we estimate expected gradient?

SGD: Stochastic Gradient Descent (or Ascent)

- "True" gradient:
  \[
  \nabla \ell(w) = E_x [\nabla \ell(w, x)]
  \]

- Sample based approximation: take iid samples \( x^{(i)} \)
  \[
  \nabla \ell(w) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla \ell(w, x^{(i)})
  \]

  - What if we estimate gradient with just one sample???
    - Unbiased estimate of gradient
    - Very noisy!
    - Called stochastic gradient descent
      - Among many other names
    - VERY useful in practice!!!
Perceptron & Stochastic Gradient descent

**Perceptron update:**

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

**Batch hinge minimization update:**

\[ w^{(t+1)} \leftarrow w^{(t)} + \frac{1}{N} \sum_{i=1}^{N} \left\{ \mathbb{I} \left[ y^{(i)} (w^{(t)} \cdot x^{(i)}) \leq 0 \right] y^{(i)} x^{(i)} \right\} \]

**Stochastic Gradient Descent:**

**general case**

- Given a stochastic function of parameters: \( f(w) = E_x [ f(w, x) ] \)
- Want to find minimum \( w^* = \min_w E_x [ f(w, x) ] \)
- Start from \( w^{(0)} \)
- Repeat until convergence:
  - Get a sample data point \( x^{(i)} \)
  - Update parameters:
    \[ w^{(t+1)} \leftarrow w^{(t)} - \eta_t \alpha f(w^{(t)}, x^{(i)}) \]
    - \( \eta_t \) decreases with iterations
    - Complexity of gradient computation is constant in number of examples!
- Works on the online learning setting!
- 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:
  \[ E_x [\ell(w, x)] = E_x [\ln P(y|x, w) - \lambda ||w||^2] \]

- Batch gradient ascent updates:
  \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \frac{1}{N} \sum_{j=1}^{N} x_j^{(i)} y_j^{(i)} - P(Y = 1|x_j^{(i)}, w^{(t)}) \right\} \]

- Stochastic gradient ascent updates:
  - Online setting:
    \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} y_i^{(t)} - P(Y = 1|x_i^{(t)}, w^{(t)}) \right\} \]