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:
    - What is $\Delta$?
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: $x^{(t)} \leftarrow \text{user, page, ad features}$
  - Make prediction: $\hat{y}^{(t)} = \text{Sign} (w^{(t)} \cdot x^{(t)})$
  - Observe true class: $y^{(t)} \leftarrow \text{true label}$
  - Update model:
    - If prediction is not equal to truth, make a mistake
      $w^{(t+1)} \leftarrow w^{(t)} + y^{(t)} x^{(t)}$
      \[ e^{(t)}; \quad w^{(t+1)} \leftarrow w^{(t)} + y^{(t)} x^{(t)} \]

What if the data is not linearly separable?

Use features of features of features of features....

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

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:
  \[
  y = \text{Sign} \left( w^{(i)} \cdot x \right)
  \]

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

- When using high dimensional features:
  \[
  \text{Sign} \left( \sum_{i \in M^{(i)}} y^{(i)} \phi(x^{(i)}) \cdot \phi(x) \right) \text{ is the dot product between } x \text{ and mistake } i
  \]
Dot-product of polynomials \( \phi(u) \cdot \phi(v) = \) polynomials of degree exactly \( d \)

\[
\phi(u) \cdot \phi(v) = \begin{pmatrix} u_1 \\ \vdots \\ u_d \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ \vdots \\ v_d \end{pmatrix} = u_1v_1 + u_2v_2 = u \cdot v
\]

\[
\phi(u) \cdot \phi(v) = \begin{pmatrix} u_1^2 \\ u_1u_2 \\ u_1u_3 \\ \vdots \\ \vdots \\ u_1u_d \end{pmatrix} \cdot \begin{pmatrix} v_1^2 \\ v_1v_2 \\ v_1v_3 \\ \vdots \\ \vdots \\ v_1v_d \end{pmatrix} = u_1^2v_1 + 2u_1u_2v_1v_2 + u_2^2v_2 = (u \cdot v)^2
\]

Proof by one step of induction

If \( \phi(u) \) is poly of degree exactly \( d \), compute in the

\[\phi(u) \cdot \phi(v) = (u \cdot v)^2\]

Finally the Kernel Trick!!!

( Kernelized Perceptron

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

- Kernelized Perceptron prediction for \( x \):

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

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

- How about all monomials of degree up to $d$?
  - Solution 0:
  - Better solution:

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

---

Choice can make a huge difference!!

![Graph showing different weight vector representations over epochs](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????

Perceptron Prediction: Margin of Confidence
Hinge Loss

- Perceptron prediction:

- Makes a mistake when:

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

Minimizing hinge loss in Batch Setting

- Given a dataset:

  - Minimize average hinge loss:

  - How do we compute the gradient?
Subgradients of Convex Functions

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

Subgradient of Hinge

- Hinge loss:
  - Subgradient of hinge loss:
    - If $y^{(i)} (w \cdot x^{(i)}) > 0$:
    - If $y^{(i)} (w \cdot x^{(i)}) < 0$:
    - If $y^{(i)} (w \cdot x^{(i)}) = 0$:
    - In one line:
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...
  - 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:
- Want to minimize:
- Subgradient descent works the same as gradient descent:
  - But if there are multiple subgradients at a point, just pick (any) one:
Perceptron Revisited

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

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

- 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,…
  - We often minimize loss in training data:
    \[ \ell_D(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\mathbf{w}, \mathbf{x}^{(i)}) \]

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

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

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

- Taking the gradient:

- "True" gradient descent rule:

- 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:

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

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

**Stochastic Gradient Descent: general case**

- Given a stochastic function of parameters:
  - Want to find minimum

- Start from \( w^{(0)} \)
- Repeat until convergence:
  - Get a sample data point \( x^{(i)} \)
  - Update parameters:

- Works on the online learning setting!
- Complexity of gradient computation is constant in number of examples!
- In general, step size changes with iterations
### Stochastic Gradient Ascent for Logistic Regression

- Logistic loss as a stochastic function:
  \[
  E_x \left[ \ell(w, x) \right] = E_x \left[ \ln P(y|x, w) - \lambda ||w||^2 \right]
  \]

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

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

### Convergence rate of SGD

- **Theorem:**
  - (see Nemirovski et al ’09 from readings)
  - Let \( f \) be a strongly convex stochastic function
  - Assume gradient of \( f \) is Lipschitz continuous and bounded
  - Then, for step sizes:
    - The expected loss decreases as \( O(1/t) \):
Convergence rates for gradient descent/ascent versus SGD

- Number of Iterations to get to accuracy
  \[ \ell(w^*) - \ell(w) \leq \epsilon \]
- Gradient descent:
  - If func is strongly convex: \(O(\ln(1/\epsilon))\) iterations
- Stochastic gradient descent:
  - If func is strongly convex: \(O(1/\epsilon)\) iterations
- Seems exponentially worse, but much more subtle:
  - Total running time, e.g., for logistic regression:
    - Gradient descent:
    - SGD:
      - SGD can win when we have a lot of data
  - And, when analyzing true error, situation even more subtle… expected running time about the same, see readings

What you need to know

- Perceptron is optimizing hinge loss
- Subgradients and hinge loss
- (Sub)gradient decent for hinge objective
- Objective functions in ML as expectations
- Gradient estimation, rather than objective estimation
- Stochastic gradient descent -> estimate gradient from single training example
  - Mini-batches possible and useful
- Stochastic gradient ascent for logistic regression
- Analysis of stochastic gradient descent
  - Decreasing step size fundamental here
- Comparing analysis of stochastic gradient descent with gradient descent