The Perceptron Algorithm

[Rosenblatt '58, '62]

- Classification setting: $y$ in $\{-1,+1\}$
- Linear model
  - Prediction: $\hat{y} = \text{Sign} (\mathbf{w} \cdot \mathbf{x})$
- Training:
  - Initialize weight vector: $\mathbf{w}^{(0)} = 0$ or something small
  - At each time step:
    - Observe features: $\mathbf{x}^{(i)}$
    - Make prediction: $\hat{y}^{(i)} = \text{Sign} (\mathbf{w}^{(i)} \cdot \mathbf{x}^{(i)})$
    - Observe true class: $y^{(i)}$
    - Update model: $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + y^{(i)} \mathbf{x}^{(i)}$
      - If prediction is not equal to truth, if make a mistake!

E.g., $y^{(i)} = +1$
$\mathbf{w}^{(i)} \cdot \mathbf{x}^{(i)} < 0$
but $|\mathbf{w}^{(i)}| > 0$
what $\mathbf{w}$ may $\mathbf{w}^{(i)}$?

$\mathbf{w}^{(i)} \cdot \mathbf{x}^{(i)}$

by adding $\mathbf{x}^{(i)}$ to $\mathbf{w}^{(i)}$
I increase $\mathbf{w}^{(i)} \cdot \mathbf{x}^{(i)}$

At most

Similarly when
$y^{(i)} = -1$
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:
  \[ F(w') \geq F(w) + \nabla F(w)^T (w' - w) \]

- Gradients are unique at \( w \) iff function differentiable at \( w \)

- Subgradients: Generalize gradients to non-differentiable points:
  - Any plane that lower bounds function:
    \[ V \in \partial F(w) \text{ subgradient} \]
    \[ F(w') \geq F(w) + V(w' - w) \]
    For \( |w| \to 0 \):
    \[ V \in [-1, 1] \]

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

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

- Difference?
What you need to know

- Perceptron is optimizing hinge loss
- Subgradients and hinge loss
- (Sub)gradient decent for hinge objective

Kernels

Machine Learning – CSE446
Carlos Guestrin
University of Washington

May 1, 2013
Linear Separability: More formally, Using Margin

- Data linearly separable, if there exists
  - a vector \( \exists w \), \( \|w\| = 1 \)
  - a margin \( y > 0 \)
  - all points \( s.t. \ y \omega \cdot x > y \)
  - \( w \cdot x \leq y \) for \( y < 0 \)
  - \( y(x) \omega \cdot x > y \) linearly separable, margin \( y \)

Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples: \( \langle (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(T)}, y^{(T)}) \rangle \)
  - Each feature vector has bounded norm:
    \( \exists w \), \( \|w\| = 1 \)
  - If dataset is linearly separable:
    \( \forall t \ y(x^{(t)}) w \cdot x^{(t)} > y \), for \( y > 0 \)
  - Then the number of mistakes made by the online perceptron on this sequence is bounded by
    \[
    \left( \frac{c}{y} \right)^2 \leq K \]
    \( K \) is a constant, independent of \( T \) or dimensionality of \( x \).
Beyond Linearly Separable Case

- **Perceptron algorithm is super cool!**
  - No assumption about data distribution!
    - Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it’s done for ever!
    - Even if you see infinite data

- However, real world not linearly separable
  - Can’t expect never to make mistakes again
  - Analysis extends to non-linearly separable case
  - Very similar bound, see Freund & Schapire
  - Converges, but ultimately may not give good accuracy (make many many mistakes)

What if the data is not linearly separable?

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

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

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

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

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

- \(m = 100\), \(d = 6\):
  - About 1.6 billion terms

Perceptron Revisited

- Given weight vector \(w^{(i)}\), predict point \(x\) by:
  \[
  n\mathbf{m}
  \]

- 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:

- Prediction rule now:

- When using high dimensional features:
Dot-product of polynomials

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

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

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

- Sigmoid
  \[ K(u, v) = \tanh(\eta u \cdot v + \nu) \]
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