Kernelized Perceptron
Support Vector Machines

What is the perceptron optimizing?
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: $w(0) = 0$ (or smarter)
  - At each time step:
    - Observe features: $x_t$
    - Make prediction: $\hat{y}_t = \text{sign}(w^{(t)} \cdot x_t)$
    - Observe true class: $y_t$
    - Update model:
      - If prediction is not equal to truth
        $$\begin{cases} 
        w^{(t+1)} = w^{(t)} + y_t x_t & \text{if } \hat{y}_t \neq y_t \\
        w^{(t+1)} = w^{(t)} & \text{else}
        \end{cases}$$

Perceptron prediction: Margin of confidence
Hinge loss

- Perceptron prediction: $\text{sign}(\mathbf{w} \cdot \mathbf{x})$

- Makes a mistake when:

$$y(\mathbf{w} \cdot \mathbf{x}) < 0 \Rightarrow L(\mathbf{w}, \mathbf{x}) = \begin{cases} 0 & y(\mathbf{w} \cdot \mathbf{x}) = 0 \\ -y(\mathbf{w} \cdot \mathbf{x}) & \text{o.w.} \end{cases} \Rightarrow (-y(\mathbf{w} \cdot \mathbf{x}))_+$$

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

Minimizing hinge loss in batch setting

- Given a dataset: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

- Minimize average hinge loss:

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{w}, \mathbf{x}_i) = (-y_i(\mathbf{w} \cdot \mathbf{x}_i))_+$$

- How do we compute the gradient?

$$\nabla_w L(\mathbf{w}, \mathbf{x}) = -y \mathbf{x}$$

$$\nabla_w L(\mathbf{w}, \mathbf{x}) = 0$$

(no gradient here)
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:
Subgradient descent for hinge minimization

- Given data: \((x_1, y_1), \ldots, (x_N, y_N)\)

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

- 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:
  \[
  w^{(t+1)} \leftarrow w^{(t)} + \mathbb{I} \left[ y_t (w^{(t)} \cdot x_t) \leq 0 \right] y_t x_t
  \]

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

- Difference?
The kernelized perceptron

What if the data are not linearly separable?

Use features of features of features of features...

\[ \phi(x) : \mathbb{R}^d \rightarrow F \]

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

\[
\# \text{ terms} = \binom{p + d - 1}{p} = \frac{(p + d - 1)!}{p!(d - 1)!}
\]

d – input dimension
p – degree of polynomial

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

Perceptron revisited

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

• Mistake at time \( t \): \( w^{(t+1)} \leftarrow w^{(t)} + y_t x_t \)

• 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

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

Finally, the kernel trick!
Kernelized perceptron

- Every time you make a mistake, remember \((x_t, y_t)\)

- Kernelized perceptron prediction for \(x\):

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

\[
= \sum_{i \in M^{(t)}} y_i k(x_i, x)
\]
Polynomial kernels

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

• How about all monomials of degree up to $p$?
  – Solution 0:
  – Better solution:

Common kernels

• Polynomials of degree exactly $p$
  $$K(u, v) = (u \cdot v)^p$$

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

• 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

- Linear separability in higher-dim feature space
- The kernel trick
- Kernelized perceptron
- Derive polynomial kernel
- Common kernels

Support vector machines (SVMs)
Linear classifiers—Which line is better?

Pick the one with the largest margin!

$$w \cdot x + w_0 = 0$$

“confidence” = $y_i(w \cdot x_i + w_0)$
Maximize the margin

\[
\max_{\gamma, \mathbf{w}, w_0} \gamma \quad \text{subject to } y_i (\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq \gamma, \forall i \in \{1, \ldots, N\}
\]

But there are many planes...
Review: Normal to a plane

\[ \mathbf{x}_i = \bar{x}_i + \lambda \frac{\mathbf{w}}{||\mathbf{w}||} \]

A convention: Normalized margin

Canonical hyperplanes

\[ \mathbf{x}_i = \bar{x}_i + \lambda \frac{\mathbf{w}}{||\mathbf{w}||} \]
Margin maximization using canonical hyperplanes

Unnormalized problem:

$$\max_{\gamma,w,w_0} \gamma \quad \gamma \geq 1, \forall i \in \{1, \ldots, N\}$$

Normalized Problem:

$$\min_{w,w_0} \|w\|^2 \quad y_i(w \cdot x_i + w_0) \geq 1, \forall i \in \{1, \ldots, N\}$$

Support vector machines (SVMs)

$$\min_{w,w_0} \|w\|^2 \quad y_i(w \cdot x_i + w_0) \geq 1, \forall i \in \{1, \ldots, N\}$$

- Solve efficiently by many methods, e.g.,
  - quadratic programming (QP)
  - Well-studied solution algorithms
  - Stochastic gradient descent
- Hyperplane defined by support vectors
What if data are not linearly separable?

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

What if data are still not linearly separable?

\[
\min_{\mathbf{w}, w_0} \|\mathbf{w}\|_2^2
\]

\[y_i (\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1, \forall i \in \{1, \ldots, N\}\]

- If data are not linearly separable, some points don’t satisfy margin constraint:
  - How bad is the violation?
  - Tradeoff margin violation with \(\|\mathbf{w}\|\):
SVMs for non-linearly separable data, meet my friend the Perceptron...

- Perceptron was minimizing the hinge loss:
  \[ \sum_{i=1}^{N} (1 - y_i(w \cdot x_i + w_0))_+ \]
- SVMs minimizes the regularized hinge loss!!
  \[ ||w||^2 + C \sum_{i=1}^{N} (1 - y_i(w \cdot x_i + w_0))_+ \]

Stochastic gradient descent for SVMs

- Perceptron minimization:
  \[ \sum_{i=1}^{N} (1 - y_i(w \cdot x_i + w_0))_+ \]
- SGD for Perceptron:
  \[ w^{(t+1)} \leftarrow w^{(t)} + \lambda \left[ y_l(w^{(t)} \cdot x_l) \leq 0 \right] y_l x_l \]
- SVMs minimization:
  \[ ||w||^2 + C \sum_{i=1}^{N} (1 - y_i(w \cdot x_i + w_0))_+ \]
- SGD for SVMs:
Mixture model example

\[ C = 10000 \]

\[ C = 0.01 \]

From Hastie, Tibshirani, Friedman book

---

Mixture model example – kernels

SVM - Degree-4 Polynomial in Feature Space

SVM - Radial Kernel in Feature Space

From Hastie, Tibshirani, Friedman book
What you need to know…

• Maximizing margin
• Derivation of SVM formulation
• Non-linearly separable case
  – Hinge loss
  – a.k.a. adding slack variables
• SVMs = Perceptron + L₂ regularization
• Can optimize SVMs with SGD
  – Many other approaches possible