CSDS 440: Machine Learning

Soumya Ray (he/him, sray@case.edu)

Olin 516

Office hours T, Th 11:15-11:45 or by appointment

Support Vector Machines

- Combines three fundamental ideas
  - Linear discriminants
  - Margins
  - Kernels

- A theoretically well justified and empirically well-behaved method arising from three fields: ML (Cortes & Vapnik), Statistics (Wahba), Operations Research (Mangasarian)

The primal (separable) SVM

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{so that } \forall i, -\left[y_i(\mathbf{w} \bullet \mathbf{x}_i + b) - 1\right] \le 0$$

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{so that } \forall i, -\left[y_i(\mathbf{w} \bullet \varphi(\mathbf{x}_i) + b) - 1\right] \le 0$$

The primal (separable) SVM

$$\min_{\mathbf{w},b} \frac{1}{2}\left\|\mathbf{w}\right\|^2$$

$$\text{so that } \forall i, -\left[y_i(\mathbf{w}\bullet\mathbf{x}_i + b) - 1\right] \le 0$$

$$\therefore \ell(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\left\|\mathbf{w}\right\|^2 - \sum_i \alpha_i \left[y_i(\mathbf{w}\bullet\mathbf{x}_i + b) - 1\right]$$

Linearly-separable SVM, Dual Form

$$\max_{\alpha} D(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j \mathbf{x}_i \bullet \mathbf{x}_j$$

$$\text{so that } \boldsymbol{\alpha} \geq 0, \sum_i \alpha_i y_i = 0$$

From derivative w.r.t b

Kernels

- Notice that by using the dual formulation, we could write the formulation and the solution in terms of $\mathbf{x}_i \cdot \mathbf{x}_j$
  – In general, $\boldsymbol{\varphi}(\mathbf{x}_i) \cdot \boldsymbol{\varphi}(\mathbf{x}_j)$

- Define the kernel to be $K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i) \cdot \boldsymbol{\varphi}(\mathbf{x}_j)$

- For m examples, get an m x m matrix --- the kernel matrix

Nonlinear SVM, kernelized dual form

$$\max_{\alpha} D(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{so that } \alpha \geq 0, \sum_i \alpha_i y_i = 0$$

Why is this useful?

- Given φ, easy to find K
- More interestingly, for certain functions K, can show *there must exist* a φ for which K is a kernel
- This φ could be very high dimensional, but the kernel computation is much more efficient
- This allows us to do classification in very high dimensional spaces, without ever "mapping" the data into those spaces

"Kernel trick"

Example

$$K(\mathbf{a},\mathbf{b}) = (\mathbf{a}\bullet\mathbf{b})^2$$

$O(n)$ computation

$$= \left(\sum_i a_i b_i\right)\left(\sum_j a_j b_j\right)$$

$$= \sum_{i,j} (a_i a_j)(b_i b_j)$$

$$= \varphi(\mathbf{a})\bullet\varphi(\mathbf{b}), \text{ where}$$

$O(n^2)$ computation!

$$\varphi(\mathbf{x}) = [x_1^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_ix_j, \ldots, x_n^2]$$

Example

$$\mathbf{a} = (1,2), \mathbf{b} = (3,4)$$

$$K(\mathbf{a},\mathbf{b}) = (\mathbf{a} \bullet \mathbf{b})^2 = (1 \times 3 + 2 \times 4)^2 = 121$$

$$\text{If } K(\mathbf{a},\mathbf{b}) = \varphi(\mathbf{a})\varphi(\mathbf{b}) = (\mathbf{a} \bullet \mathbf{b})^2 \text{ then}$$

$$\varphi(\mathbf{a}) = [a_1^2, \sqrt{2}a_1 a_2, a_2^2] = [1, 2\sqrt{2}, 4]$$

$$\varphi(\mathbf{b}) = [b_1^2, \sqrt{2}b_1 b_2, b_2^2] = [9, 12\sqrt{2}, 16]$$

$$\varphi(\mathbf{a})\varphi(\mathbf{b}) = 9 + 48 + 64 = 121$$

What is a valid kernel?

- Intuitively, the dot product measures the (unnormalized) cosine of the angle between two vectors
  - A measure of similarity
  - "large" $K(\mathbf{x},\mathbf{y}) \rightarrow \mathbf{x}$ and $\mathbf{y}$ are "similar"

- Suppose we choose some other function that measures similarity
  - E.g., $K(\mathbf{x},\mathbf{y}) = \exp\left(\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right)$
  - Is this a valid kernel?

Yes! Called a Gaussian or RBF Kernel. Corresponds to infinite-dimensional φ!!

Mercer's Conditions

Let $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a function. K is a valid kernel iff for all finite $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, the kernel matrix is symmetric positive semidefinite.

Necessary *and* sufficient!

Symmetry: $K(\mathbf{x}, \mathbf{y}) = K(\mathbf{y}, \mathbf{x})$

PSD ($K \geq 0$): $\forall \mathbf{v} \neq 0, \mathbf{v}^T K \mathbf{v} \geq 0$

- Given any kernel(s), can compose them in various ways to get other kernels

Classification

$$\max_{\alpha} D(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{so that } \alpha \geq 0, \sum_i \alpha_i y_i = 0$$

- Note that classification also does not require φ:

$$\mathbf{w} \bullet \varphi\left(\mathbf{x}_{new}\right) = \sum_{i \in \text{Support Vectors}} \alpha_i y_i \varphi\left(\mathbf{x}_i\right) \bullet \varphi\left(\mathbf{x}_{new}\right)$$

$$= \sum_{i \in \text{Support Vectors}} \alpha_i y_i K\left(\mathbf{x}_i, \mathbf{x}_{new}\right)$$

Generalizing kernels

- Kernel functions can be used in many other contexts, thanks to the Representer Theorem

Representer Theorem

- *Any* optimization program of the form

$$\min_f \frac{1}{2} g\left(\|f\|\right) + C\sum_i L(y_i, f(\varphi(\mathbf{x}_i))))$$

- With g monotonic has a solution that looks like: $$f(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$$

- (Kimeldorf and Wahba, Scholkopf et al)

SVM

- Standard SVM:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$$

$$\text{so that } \forall i, \left[y_i(\mathbf{w} \bullet \mathbf{x}_i + b) + \xi_i \right] \geq 1, \xi_i \geq 0$$

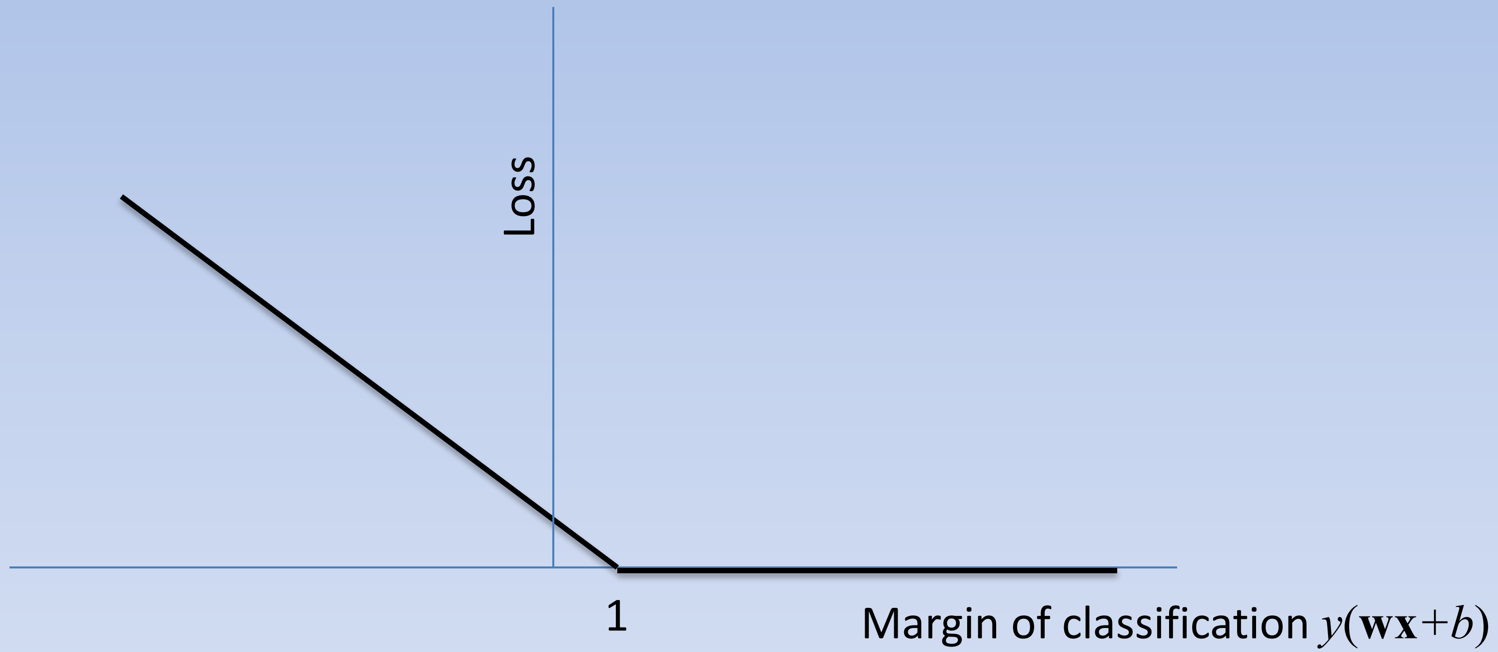
$$\text{So } \xi_i = \left(1 - y_i(\mathbf{w} \bullet \mathbf{x}_i + b)\right)_+$$

Plus function (Also RELU)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \left[\left(1 - y_i(\mathbf{w} \bullet \mathbf{x}_i + b)\right)_+ \right]$$

(Hinge Loss)

“Hinge” Loss



Logistic Regression

- Let us look again at the objective function we optimize to get LR (with overfitting control):

$$\mathbf{w}, b = \arg\min \frac{1}{2}\|\mathbf{w}\|^2 + C\left[\begin{array}{l} \sum_{i \in pos} -\log\left(P(y=1 \mid \mathbf{x}_i)\right) \\ + \sum_{i \in neg} -\log\left(1 - P(y=1 \mid \mathbf{x}_i)\right) \end{array}\right]$$

Rewriting the LR objective

- Note that, if we make the class labels 1 and -1:

$$p(Y=1\mid \mathbf{x})=\frac{1}{1+e^{-(\mathbf{w}\bullet\mathbf{x}+b)}}$$

$$p(Y=-1\mid \mathbf{x})=1-\frac{1}{1+e^{-(\mathbf{w}\bullet\mathbf{x}+b)}}=\frac{e^{-(\mathbf{w}\bullet\mathbf{x}+b)}}{1+e^{-(\mathbf{w}\bullet\mathbf{x}+b)}}=\frac{1}{1+e^{(\mathbf{w}\bullet\mathbf{x}+b)}}$$

So $p(Y=y\mid \mathbf{x})=\dfrac{1}{1+e^{-y(\mathbf{w}\bullet\mathbf{x}+b)}}$, and

$$-\log p(Y=y\mid \mathbf{x})=\log\left(1+e^{-y(\mathbf{w}\bullet\mathbf{x}+b)}\right)$$

Rewriting the LR objective

- So we can rewrite the objective:

$$\mathbf{w}, b = \arg\min \frac{1}{2}\|\mathbf{w}\|^2 + C\left[\begin{array}{l}\displaystyle\sum_{i\in pos} -\log\big(P(y=1\,|\,\mathbf{x}_i)\big) \\ \displaystyle+\sum_{i\in neg} -\log\big(1-P(y=1\,|\,\mathbf{x}_i)\big)\end{array}\right]$$

$$= \arg\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \log\left(1+e^{-y_i(\mathbf{w}\bullet\mathbf{x}_i+b)}\right)$$

SVM and Logistic Regression

- Standard SVM:

Hinge Loss

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \left[\left(1 - y_i(\mathbf{w}\bullet\mathbf{x}_i + b)\right)_+\right]$$

Margin

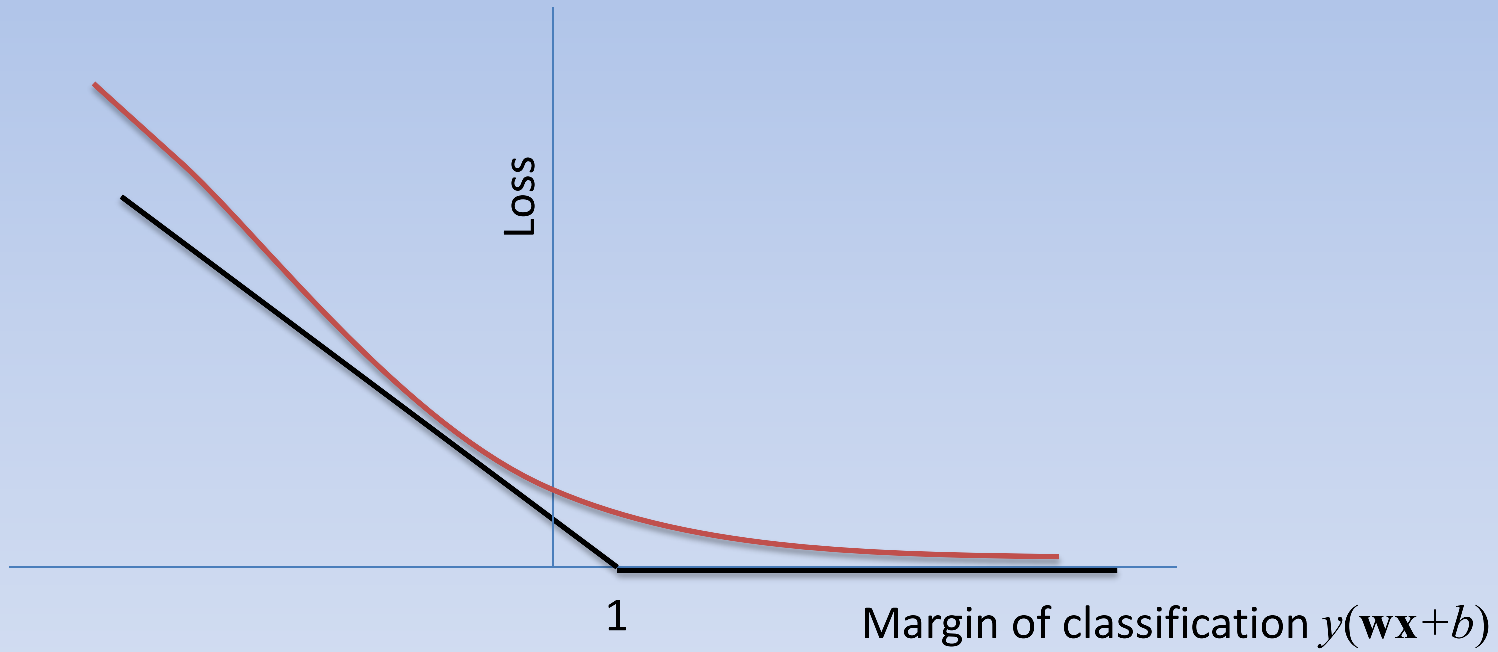
Error on training data

- (Regularized) Logistic Regression:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \log\left(1 + e^{-y_i(\mathbf{w}\bullet\mathbf{x}_i + b)}\right)$$

Logistic Loss

“Hinge” Loss vs Logistic Loss



Kernel Logistic Regression

- The LR objective "looks like" an SVM with an alternative loss function
- By the Representer theorem, the solution to this also must be $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
- And so if we introduce kernels:

$$f(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$$

Zhu and Hastie, "KLR and the Import Vector Machine" (ask for paper)

Pros and Cons of SVMs

+ Well-justified, rigorous theory combines fundamental ideas
+ "Builds" representation (connection to deep learning); can learn very complex hypotheses; has "built-in" overfitting control
+ Numerous practical applications; often very good performance
+ Kernelizing is a very powerful idea thanks to the representer theorem

− Can be sensitive to noise and outliers with nonlinear kernels
− Requires input scaling
− Not so easy to implement
− Very memory intensive due to large kernel matrices and constraints
− Not easy to parallelize/GPUize