支持向量机（下）

Machine Learning Engineer

机器学习工程师

讲师：Ivan

稀牛学院 網易 NetEase

目 录

CONTENTS

01

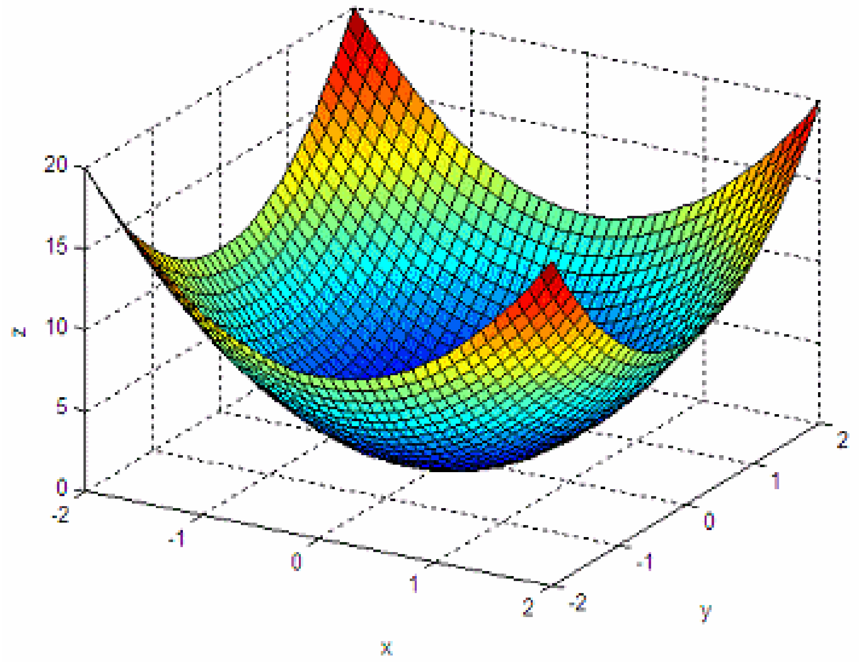
SVM 对偶形式

1.1 SVM 约束优化问题

线性SVM原问题 (Primal Problem)

$$\min \quad \frac{1}{2}|w|_2^2$$

$$s.t. \quad y_i w^T x_i \geq 1, \forall\, i \in [n]$$



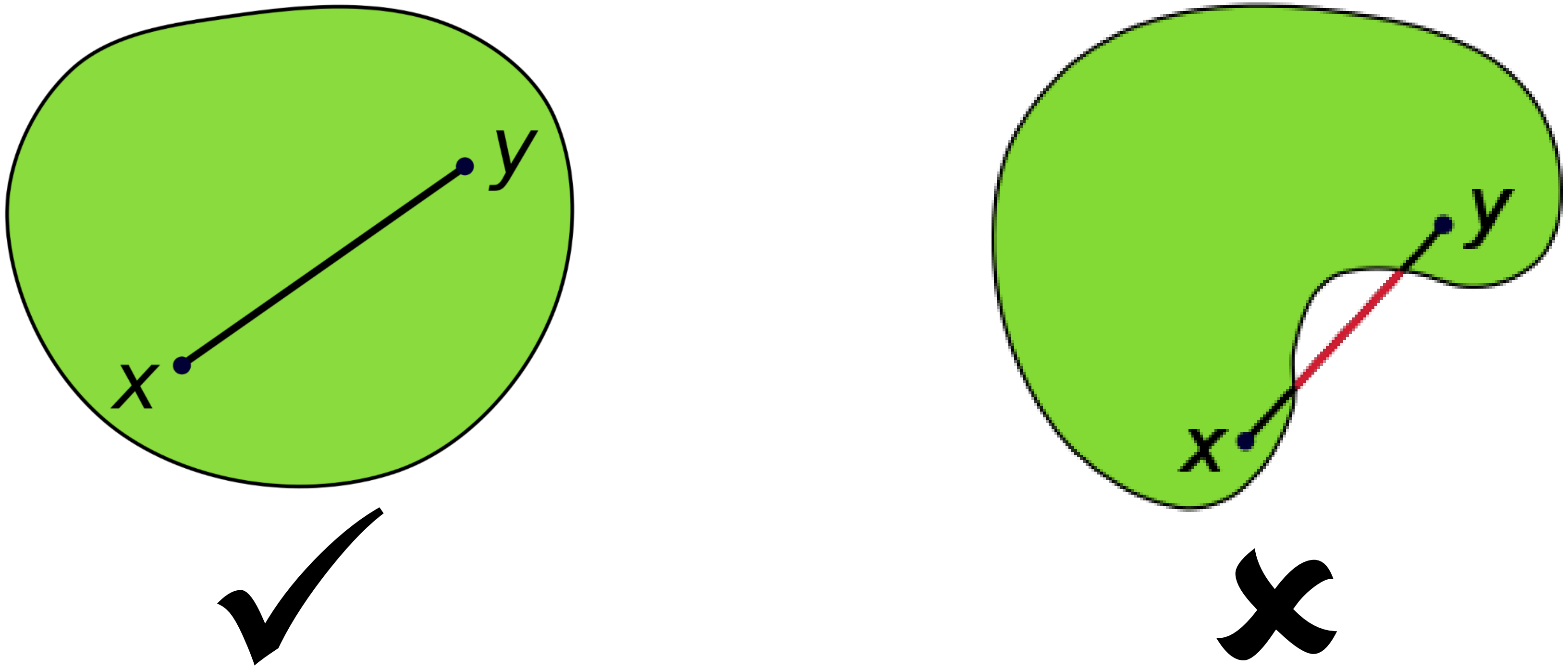
- 目标函数关于w是二次函数
- 约束条件关于w是线性函数
- 核心：二次凸优化问题 (Quadratic Programming)
    - 光滑优化函数
    - 局部最优值即全局最优值

SVM 约束优化问题

凸优化 101

- 凸集合：

For $x, x' \in X$ it follows that $\lambda x + (1-\lambda)x' \in X$ for $\lambda \in [0,1]$

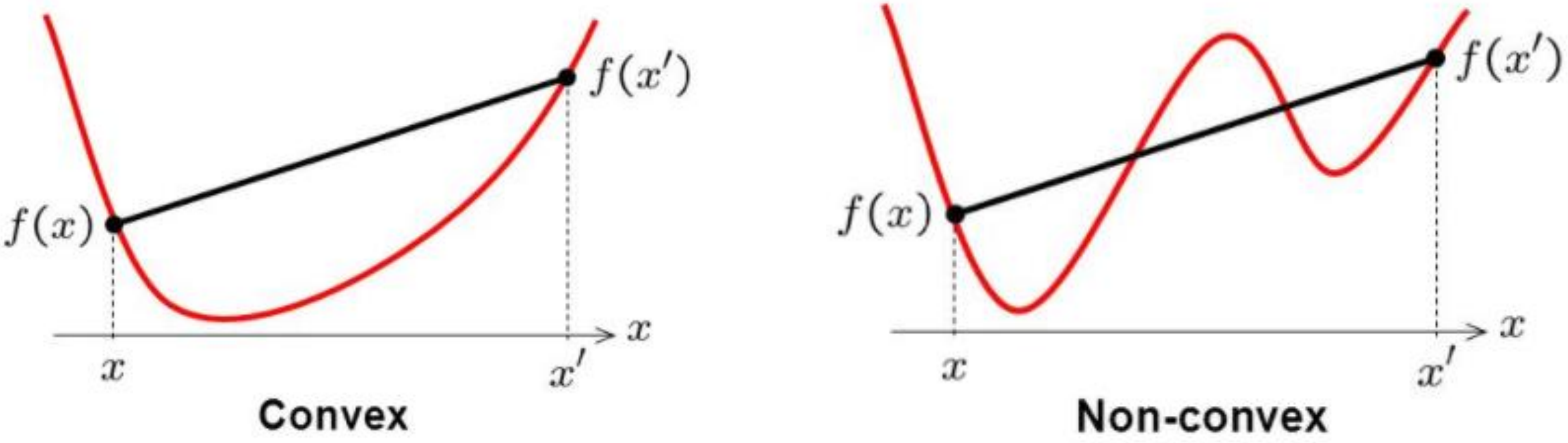


SVM 约束优化问题

凸优化 101

- 凸函数：

$$\lambda\lambda f(x) + (1-\lambda) f(x') \geq f(\lambda x + (1-\lambda) x') \text{ for } \lambda \in [0,1]$$



1.1 SVM 约束优化问题

凸优化问题一般形式：

$$\underset{x}{\text{minimize}}\ f(x)$$
$$\text{subject to } c_i(x) \leq 0 \text{ for all } i$$

- 目标函数f(x) 是凸函数
- 约束函数$c_i(x)$是凸函数

$$\min \quad \frac{1}{2}|w|_2^2$$
$$s.t. \quad y_i w^T x_i \geq 1, \forall\, i \in [n]$$

- 二次函数是凸函数
- 线性函数是凸函数

Lagrange乘子法：将约束优化转换为非约束优化

$$\underset{x}{\text{minimize}}\ f(x)$$
$$\text{subject to } c_i(x) \le 0 \text{ for all } i$$

$$\min_{x} \max_{\lambda_i \ge 0} \quad f(x) + \sum_{i=1}^{n} \lambda_i c_i(x)$$

- 为每一个不等式约束引入一个Lagrange乘子λ_i
- Lagrange乘子$\lambda_i \geq 0, \forall i \in [n]$
- 将原来的minimize问题变为一个minimax问题
- 通过Lagrange乘子法将约束优化问题转换为非约束优化问题

- 转换后的非约束优化问题与原约束优化问题最优解相同

为什么？

Lagrange乘子法：将约束优化转换为非约束优化

$$\underset{x}{\text{minimize}}\ f(x)$$

$$\text{subject to } c_i(x) \leq 0 \text{ for all } i$$

$$\Downarrow$$

$$\min_{x} \max_{\lambda_i \geq 0} \quad f(x) + \sum_{i=1}^{n} \lambda_i c_i(x)$$

推导：

1.2 Lagrange乘子法

Lagrange乘子法：将约束优化转换为非约束优化

$$\underset{x}{\text{minimize}}\ f(x)$$

$$\text{subject to } c_i(x) \leq 0 \text{ for all } i$$

$$\Downarrow$$

P(Primal): $$\min_{x} \max_{\lambda_i \geq 0} \quad f(x) + \sum_{i=1}^{n} \lambda_i c_i(x)$$

$$\Downarrow$$

D(Dual): $$\max_{\lambda_i \geq 0} \min_{x} \quad f(x) + \sum_{i=1}^{n} \lambda_i c_i(x)$$

$$P^* = D^*$$

1.2 Lagrange乘子法

Lagrange乘子法：将约束优化转换为非约束优化

Lagrange 函数: $\mathcal{L}(x,\lambda) = f(x) + \sum_{i=1}^{n} \lambda_i c_i(x)$

对偶理论(Duality Theory):

$$\min_x \max_\lambda \ \mathcal{L}(x,\lambda) = \max_\lambda \min_x \ \mathcal{L}(x,\lambda)$$

定义对偶函数：

$$g(\lambda) = \min_x \ \mathcal{L}(x,\lambda)$$

Lagrange乘子法

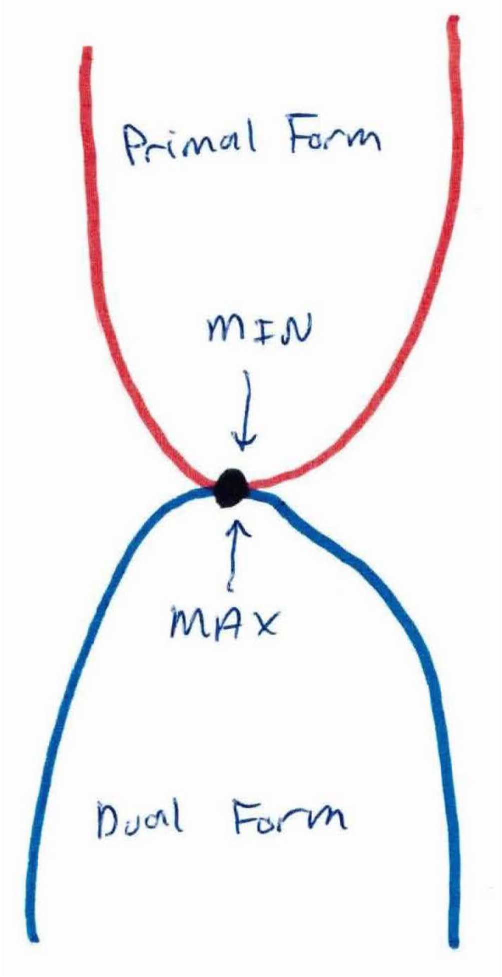
原问题与对偶问题：

对于任意可行解x, λ 我们有

$$f(x) \geq g(\lambda)$$

并且原问题与对偶问题的最优解相等: $f^*(x) = g^*(\lambda)$

对于凸优化问题(convex optimization problems)，可以通过求解对偶问题的最优解来解决原问题！



1.2 Lagrange乘子法

原问题与对偶问题：

$$\mathcal{L}(x,\lambda) = f(x) + \sum_{i=1}^{n} \lambda_i c_i(x) \quad \Longrightarrow \quad g(\lambda) = \min_x \ \mathcal{L}(x,\lambda)$$

Lagrange函数　　　　对偶函数

Primal 原问题

$$\min_x \quad f(x)$$

$$\text{s.t.} \quad c_i(x) \le 0,\ \forall i$$

$$\Longleftrightarrow$$

Dual 对偶问题

$$\max_\lambda \quad g(\lambda)$$

$$\text{s.t.} \quad \lambda_i \ge 0,\ \forall i$$

1.3 SVM 对偶形式

原问题与对偶问题：

$$\mathcal{L}(x,\lambda) = f(x) + \sum_{i=1}^{n} \lambda_i c_i(x) \quad \Longrightarrow \quad g(\lambda) = \min_{x} \ \mathcal{L}(x,\lambda)$$

Lagrange函数

对偶函数

SVM Primal

$$\min \quad \frac{1}{2}|w|_2^2$$

$$s.t. \quad y_i w^T x_i \geq 1, \forall\, i \in [n]$$

$$\Longleftrightarrow$$

SVM Dual

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j)$$

$$s.t. \quad \alpha_i \geq 0, \sum_{i=1}^{n} y_i \alpha_i = 0$$

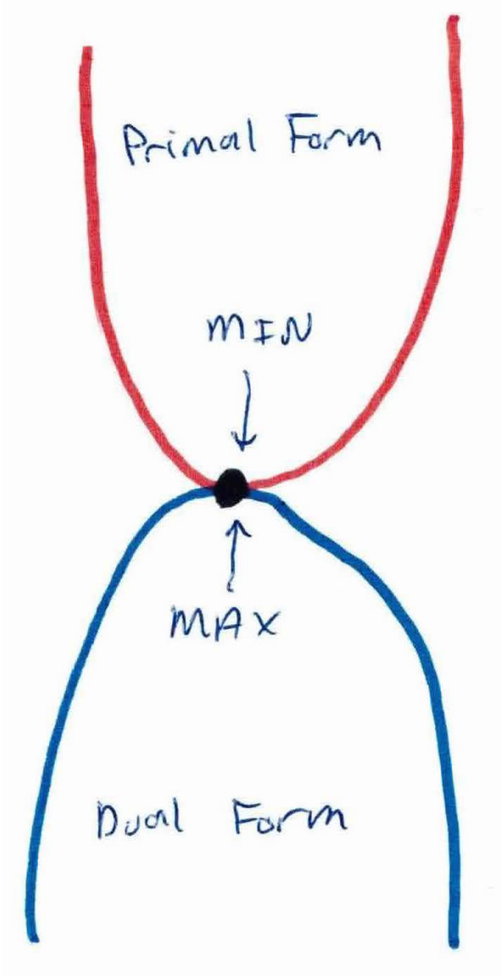
SVM 对偶形式

SVM对偶问题：

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j)$$

$$s.t. \quad \alpha_i \geq 0, \sum_{i=1}^{n} y_i \alpha_i = 0$$

- 凸二次优化问题的对偶问题还是一个凸二次优化问题
- 原问题优化变量为w
- 对偶问题优化变量为$\alpha_i, \forall i \in [n]$
- w与α_i的关系: $w = \sum_{i=1}^{n} \alpha_i y_i x_i$



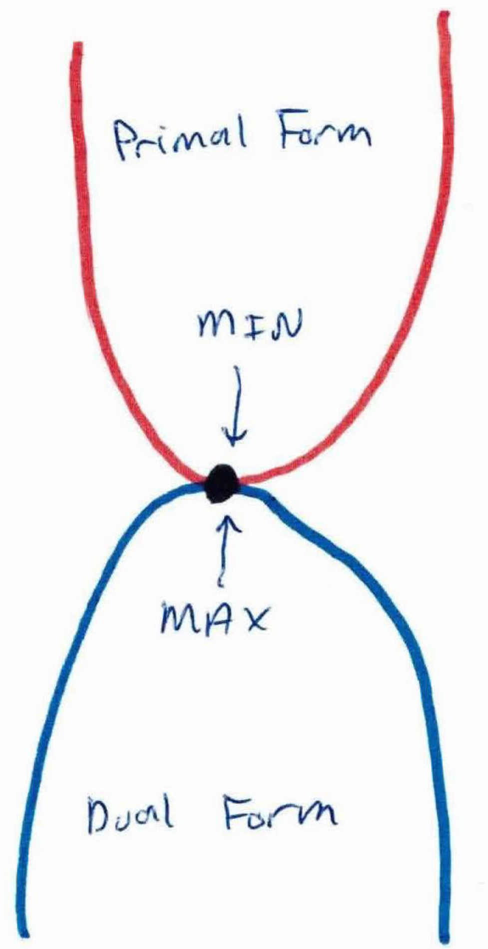
1.3 SVM 对偶形式

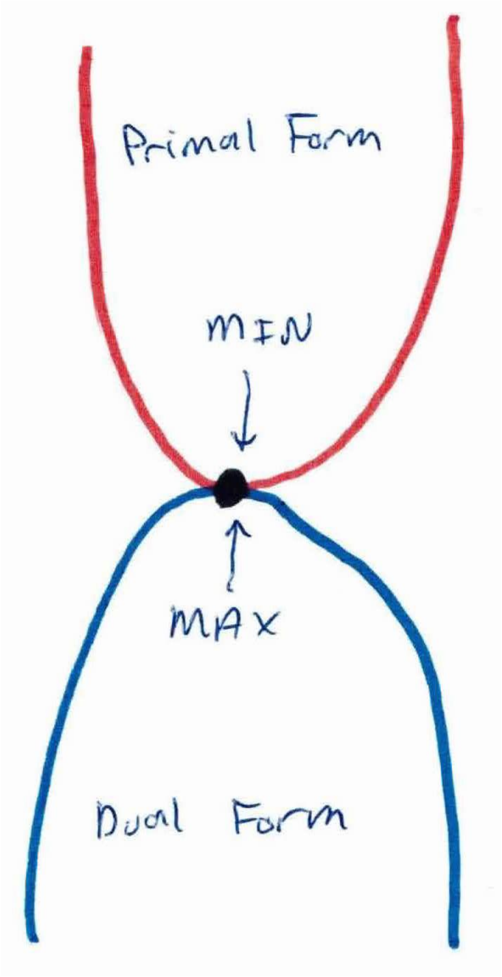
SVM对偶问题：

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j)$$

$$s.t. \quad \alpha_i \geq 0, \sum_{i=1}^{n} y_i \alpha_i = 0$$

- w与α_i的关系: $w = \sum_{i=1}^{n} \alpha_i y_i x_i$
- w只依赖于$\alpha_i > 0$的点，称为支持向量
- w通常有稀疏的表达形式（只有少量支持向量）
- 用w来进行预测：$y = w^T x = \sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x)$
- **对偶问题以及预测问题只依赖于向量内积**





要点总结

1.1 凸优化的基本概念：凸集合与凸函数

1.2 凸优化原问题以及其对偶问题

1.3 凸二次规划及其对偶形式

1.4 线性SVM的对偶形式

02 核函数以及核技巧

2.1 特征映射

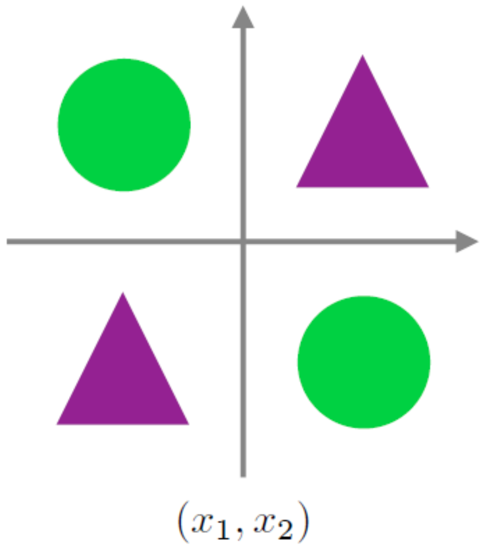
2.2 核函数(Kernel function)

2.1 特征映射

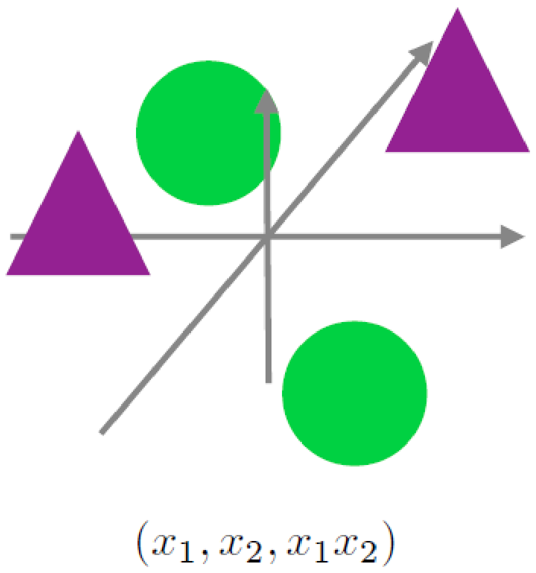
什么是特征映射?

$$\phi: R^d \rightarrow R^p, \qquad \mathrm{d} \leq p$$

将输入数据从低维空间映射到高维空间的函数变换，使得变换后的数据更加容易处理（分类/回归）



$$\phi(x_1, x_2) = (x_1, x_2, x_1 x_2)$$

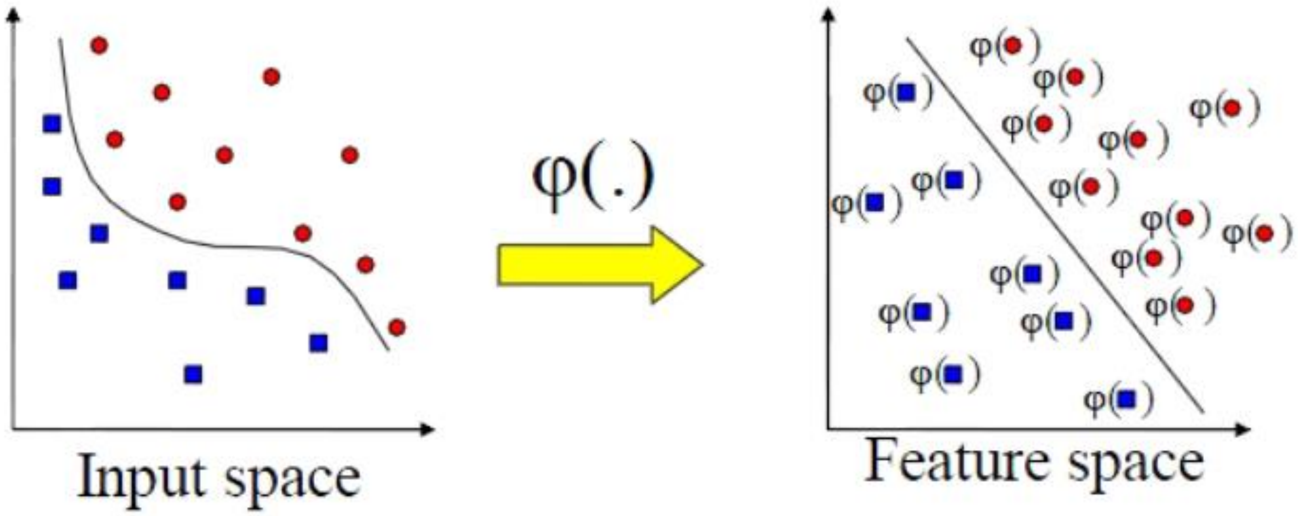


2.1 特征映射

什么是特征映射?

$$\phi: R^d \rightarrow R^p, \qquad \mathrm{d} \leq p$$

将输入数据从低维空间映射到高维空间的函数变换，使得变换后的数据更加容易处理（分类/回归）



什么是特征映射?

$$\phi: R^d \rightarrow R^p, \qquad \mathrm{d} \leq p$$

将输入数据从低维空间映射到高维空间的函数变换，使得变换后的数据更加容易处理（分类/回归）

更多例子：

- $\phi(x) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right),\ \phi(x) \cdot \phi(x') = (x_1x_1' + x_2x_2')^2 = (x \cdot x')^2$

多项式特征变换

什么是特征映射?

$$\phi: R^d \to R^p, \qquad \mathrm{d} \leq p$$

将输入数据从低维空间映射到高维空间的函数变换，使得变换后的数据更加容易处理（分类/回归）

问题：

- 如何定义适合问题的特征变换？
- 显式定义特征变换会增加计算复杂度？
- 例如：原本有1000个特征，使用二项式变换后，有500,000 (50W)个特征
- 更高维的多项式变换会引入更多的特征，极大增加计算复杂度

2.2 核函数

关键想法：我们不需要显式地计算出特征映射，只需要变换后特征的内积！

线性SVM对偶形式：

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \boxed{(x_i \cdot x_j)}$$

输入数据在原空间的内积（点积）

$$s.t. \quad \alpha_i \geq 0, \sum_{i=1}^{n} y_i \alpha_i = 0$$

如何简单地计算出：$\phi(x_i) \cdot \phi(x_j), \forall i, j \in [n]$?

- 直接计算变换后的内积，不通过$\phi(x)$
- 例如：$\phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$, $\phi(x) \cdot \phi(x') = (x_1x_1' + x_2x_2')^2 = (x \cdot x')^2$

2.2 核函数

关键想法：我们不需要显式地计算出特征映射，只需要变换后特征的内积！

定义核函数：

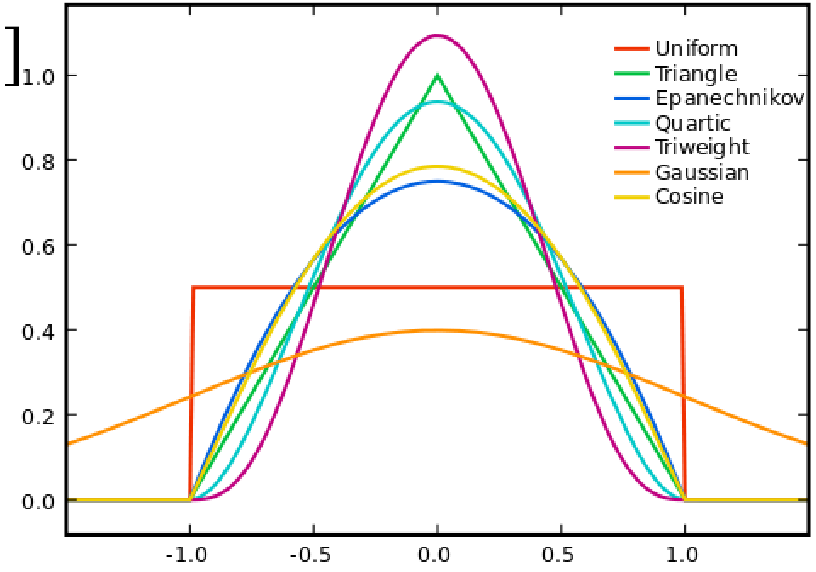
$$K(x,x'): R^d \times R^d \to R$$

- 核函数是一个对称函数：$K(x,x') = K(x',x)$
- 核函数隐式地定义了一个特征映射：$\exists\phi, s.t., K(x,x') = \phi(x)\cdot\phi(x')$
- 核函数的计算在原空间：计算复杂度低
- 例如：$K(x,x') = (x\cdot x' + c)^k$，称为$k$阶多项式核函数
    - $k = 1, c = 0$, 退化成原空间内积
    - $k = 2, c = 0$，二次多项式内积，XOR的例子
    - $k > 2, \ldots$

2.2 核函数

更多核函数例子：

- 线性核函数：$K(x,x') = x \cdot x'$
- Laplacian核函数：$K(x,x') = \exp(-\lambda|x - x'|)$
- Gaussian核函数：$K(x,x') = \exp(-\lambda|x - x'|^2)$
- 多项式核函数：$K(x,x') = (x \cdot x' + c)^k, k \in N$
- 条件密度核函数：$K(x,x') = E_c[p(x|c) \cdot p(x'|c)]$



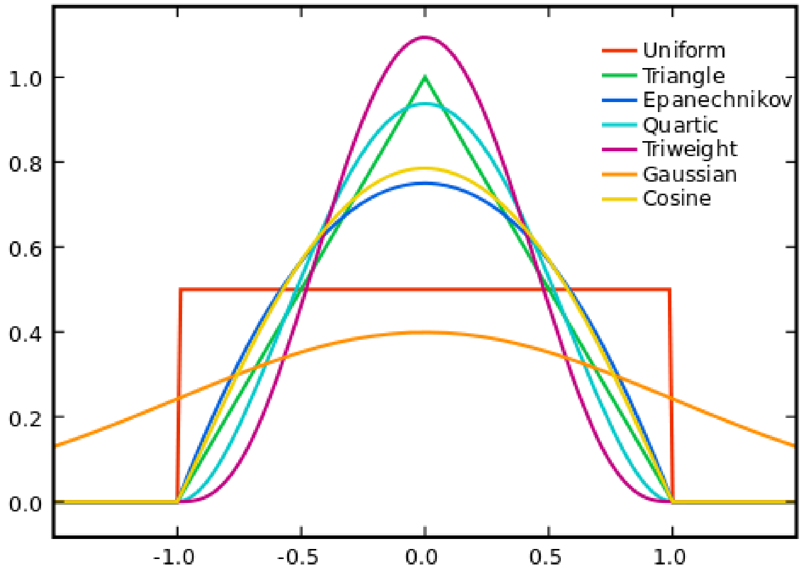
2.2 核函数

实际中经常使用的核函数：

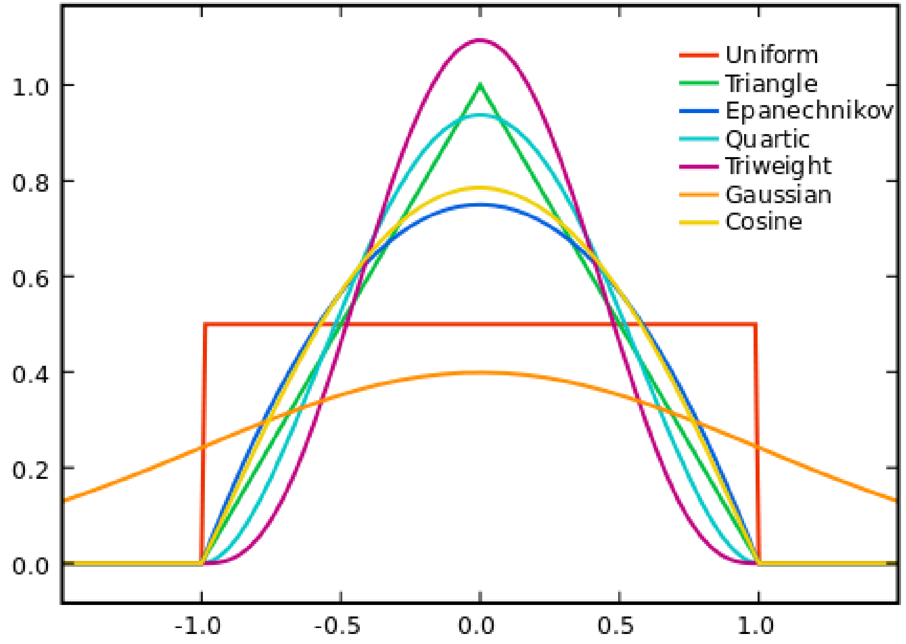
- Gaussian核函数：$K(x, x') = \exp(-\lambda |x - x'|^2)$
- 多项式核函数：$K(x, x') = (x \cdot x' + c)^k, k \in N$

- Gaussian核函数对应无穷维特征空间
- 多项式核函数对应有限维特征空间

- 工程经验：
  - Gaussian核函数：λ的选择至关重要！
  - 多项式核函数：指数k的选择至关重要！



要点总结



2.1 核函数的定义

2.2 特征变换：线性到非线性

2.3 Gaussian核函数

2.4 核技巧：将线性模型转换为非线性模型

03 非线性支持向量机

3.1 核技巧的应用

3.2 SMO 算法

如何将线性支持向量机扩展为非线性支持向量机？

SVM对偶问题：

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \boxed{(x_i \cdot x_j)}$$

输入数据在原空间的内积（点积）

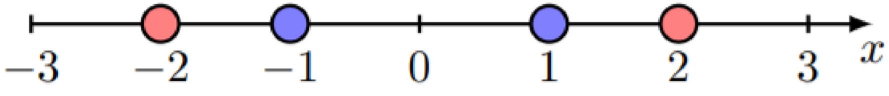
$$\Rightarrow K(x_i, x_j)$$

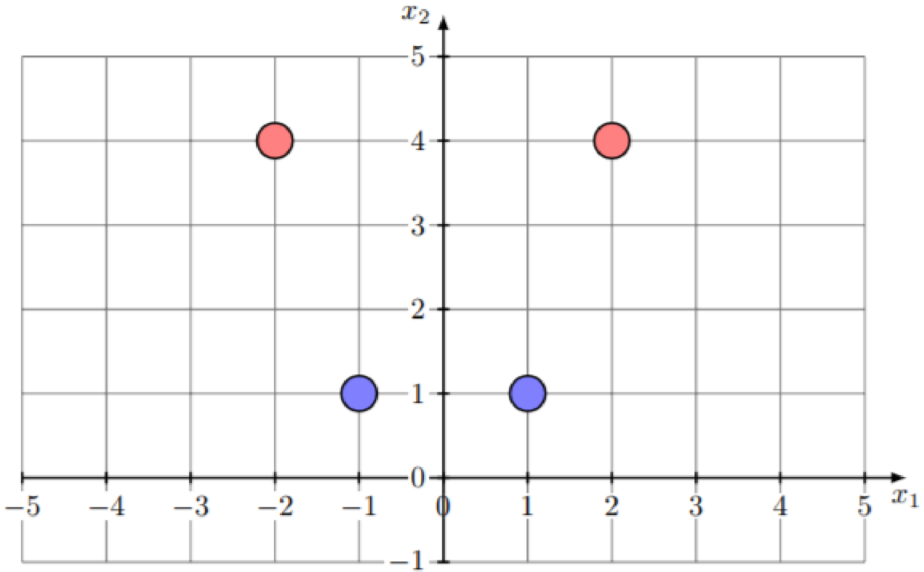
$$s.t. \quad \alpha_i \geq 0, \sum_{i=1}^{n} y_i \alpha_i = 0$$

- 将线性内积(线性核函数)替换为$K(x_i, x_j)$
- 新模型在变换后的空间仍然是线性模型
- 新模型在原空间相对于x是非线性模型
- 计算复杂度较小：只需计算核矩阵$K = K_{ij} = K(x_i, x_j)$

如何将线性支持向量机扩展为非线性支持向量机？

- 新模型在变换后的空间仍然是线性模型
- 新模型在原空间相对于x是非线性模型
- 计算复杂度较小：只需计算核矩阵$K = K_{ij} = K(x_i, x_j)$
- 例子：





非线性支持向量机优化问题：

非线性SVM对偶问题：

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i, x_j)$$

$$s.t. \quad 0 \leq \alpha_i \leq C, \qquad \sum_{i=1}^{n} y_i \alpha_i = 0$$

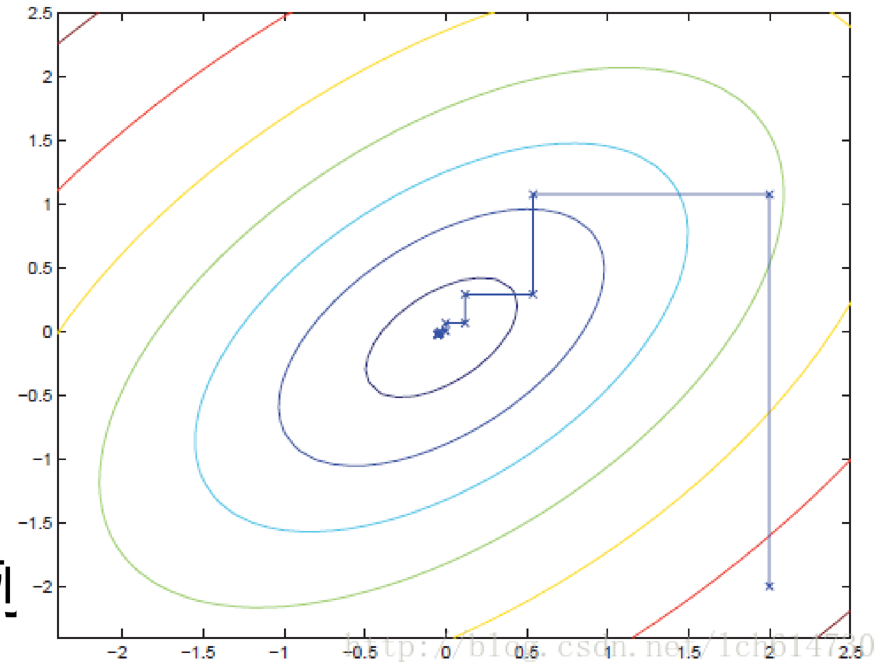
- 凸二次规划问题
- 可以求得全局最优解
- 和线性SVM对偶问题的形式几乎一样，只是将核函数换了

3.1 SMO算法

求解非线性支持向量机优化问题：

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i, x_j)$$

$$s.t. \quad 0 \le \alpha_i \le C, \qquad \sum_{i=1}^{n} y_i \alpha_i = 0$$



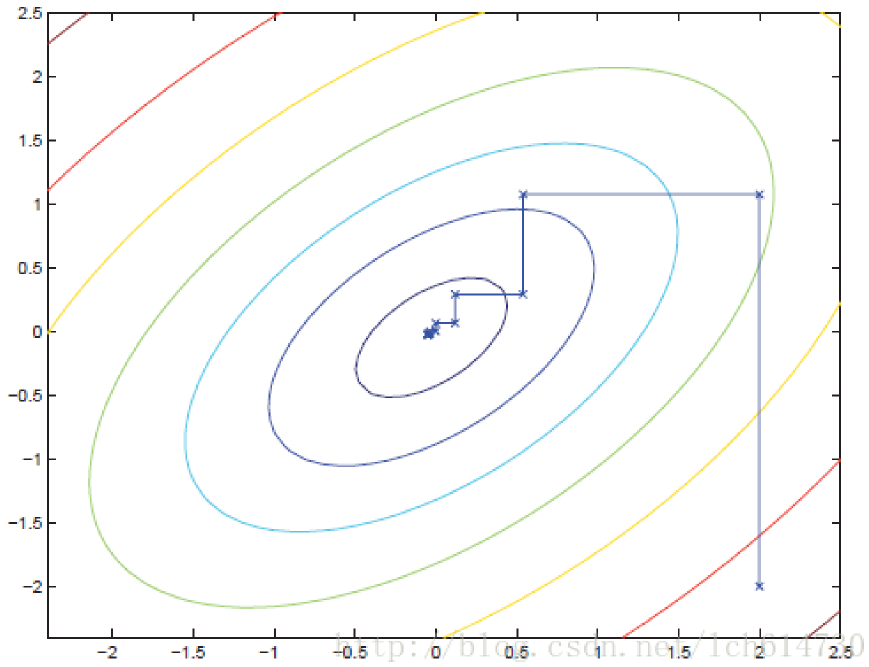
- SMO算法是Coordinate ascent算法一个特例
- 坐标上升算法：
    - 适用于光滑凸优化问题
    - 优化多个变量
    - 每次仅优化其中一个变量，固定其他所有变量不变，直至算法收敛
    - 目前SVM求解的最快算法，也是LibSVM的默认实现算法，通常远快于梯度下降算法

3.1 SMO算法

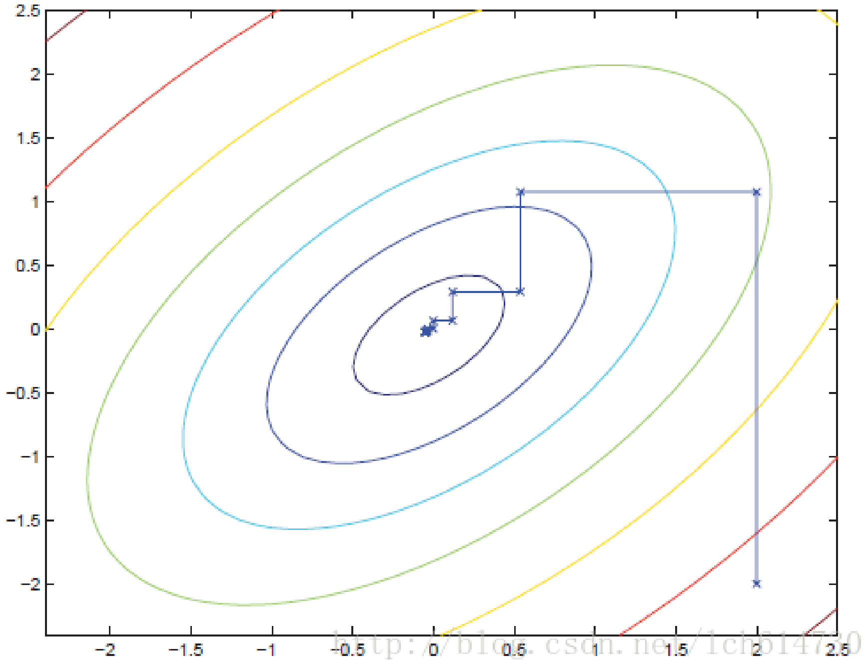
例子：

$$\min\ x_1^2 + x_2^2$$

- 初始化算法$(x_1, x_2) = (-3, -4)$
- 固定x_2，优化一维函数$x_1^2 + 16$
- 求得最优解$x_1 = 0$
- 固定$x_1 = 0$，优化一维函数$0 + x_2^2$
- 求得最优解$x_2 = 0$
- 再次固定$x_2 = 0$，优化一维函数$x_1^2 + 0$
- 求得最优解$x_1 = 0$，与上一轮迭代值相同
- 再次固定$x_1 = 0$，优化一维函数$0 + x_2^2$
- 求得最优解$x_2 = 0$，与上一轮迭代值相同
- 算法收敛，停止，得到全局最优解$(x_1, x_2) = (0,0)$



要点总结



3.1 核技巧在SVM中的应用

3.2 如何将线性支持向量机转变为非线性

3.3 SMO算法以及坐标下降算法

04

支持向量回归

回归问题的一般框架：

- 回归损失函数

$$l\big(y, f(x)\big) = \frac{1}{n}\sum_{i=1}^{n} l\big(y_i, f(x_i)\big)$$

- 正则化

$$\Omega(f) = \frac{1}{2}|w|_2^2$$

回归问题的一般框架：

- 回归约束优化问题：

$$\min \quad \frac{\lambda}{2}|w|_2^2+\frac{C}{n}\sum_{i=1}^{n} l(\epsilon_i)$$

$$\mathrm{s.}t \quad \epsilon_i = y_i - w^T x_i, \forall i \in [n]$$

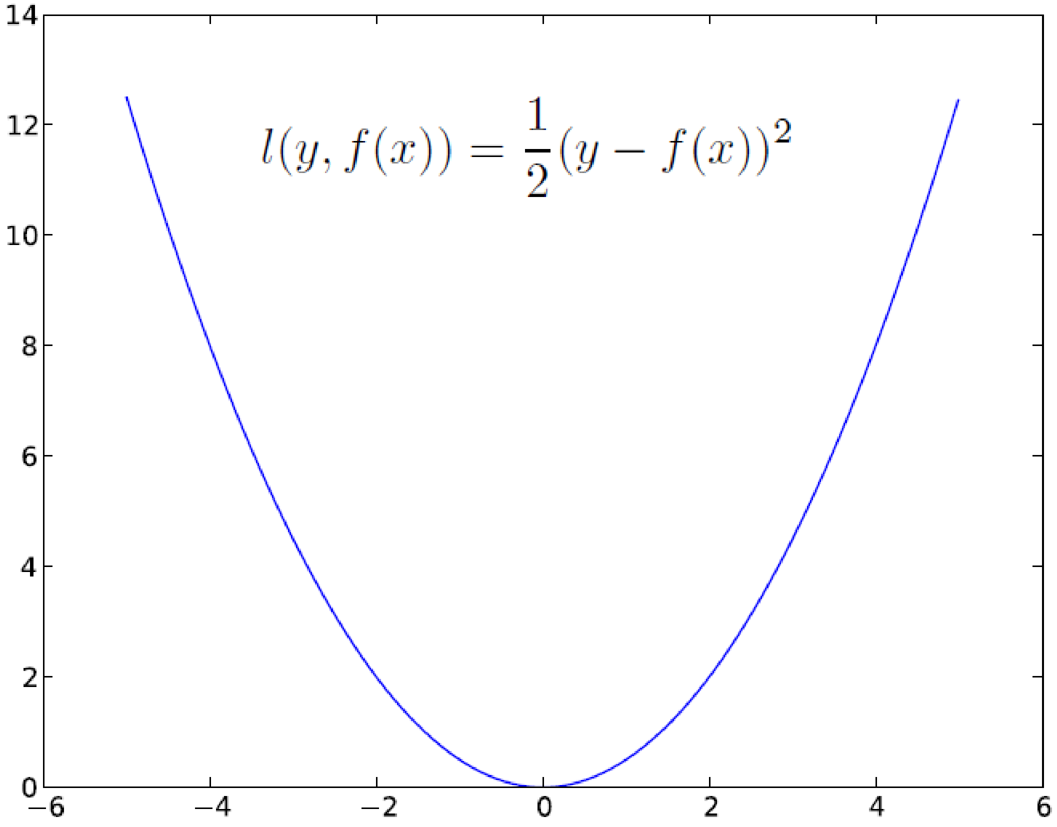
- ϵ_i被称作松弛变量
- 根据不同的损失函数会引申出不同的模型

回归问题的一般框架：

$$\min \quad \frac{\lambda}{2}|w|_2^2+\frac{C}{n}\sum_{i=1}^{n} l(\epsilon_i)$$

$$s.t \quad \epsilon_i = y_i - w^T x_i, \forall i \in [n]$$

- L2损失函数：Ridge Regression



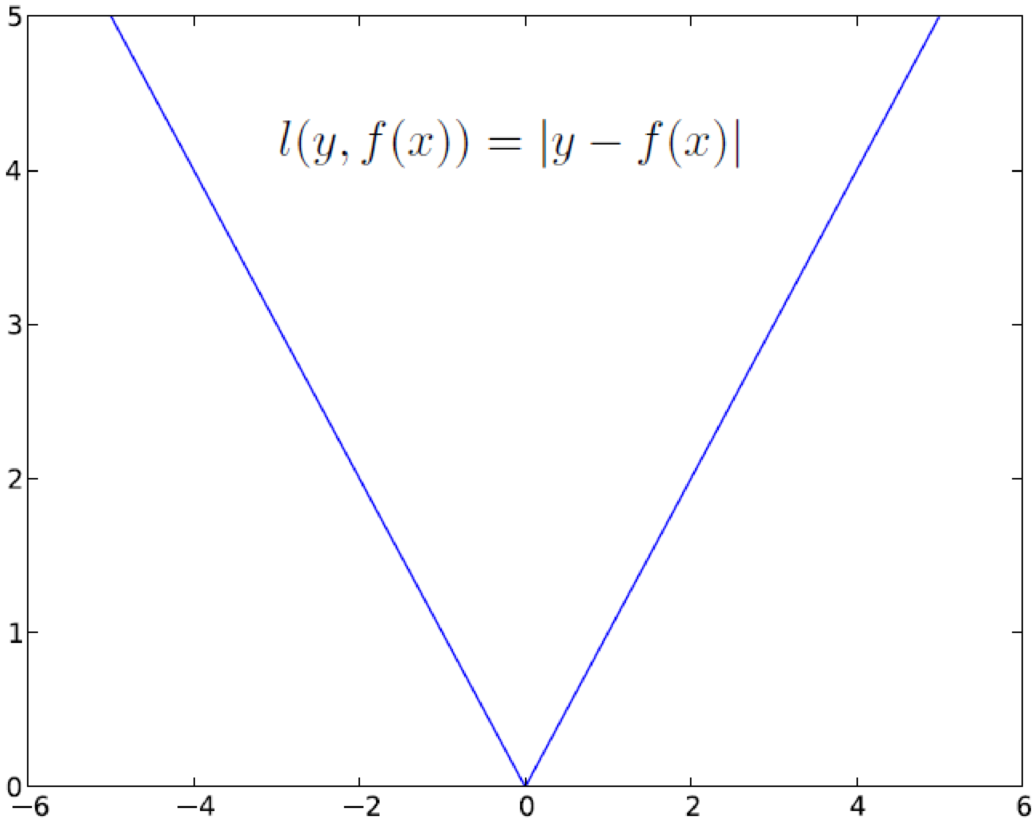
4 支持向量回归

回归问题的一般框架：

$$\min \quad \frac{\lambda}{2}|w|_2^2+\frac{C}{n}\sum_{i=1}^{n} l(\epsilon_i)$$

$$s.t \quad \epsilon_i = y_i - w^T x_i, \forall i \in [n]$$

- L1损失函数：Median Regression

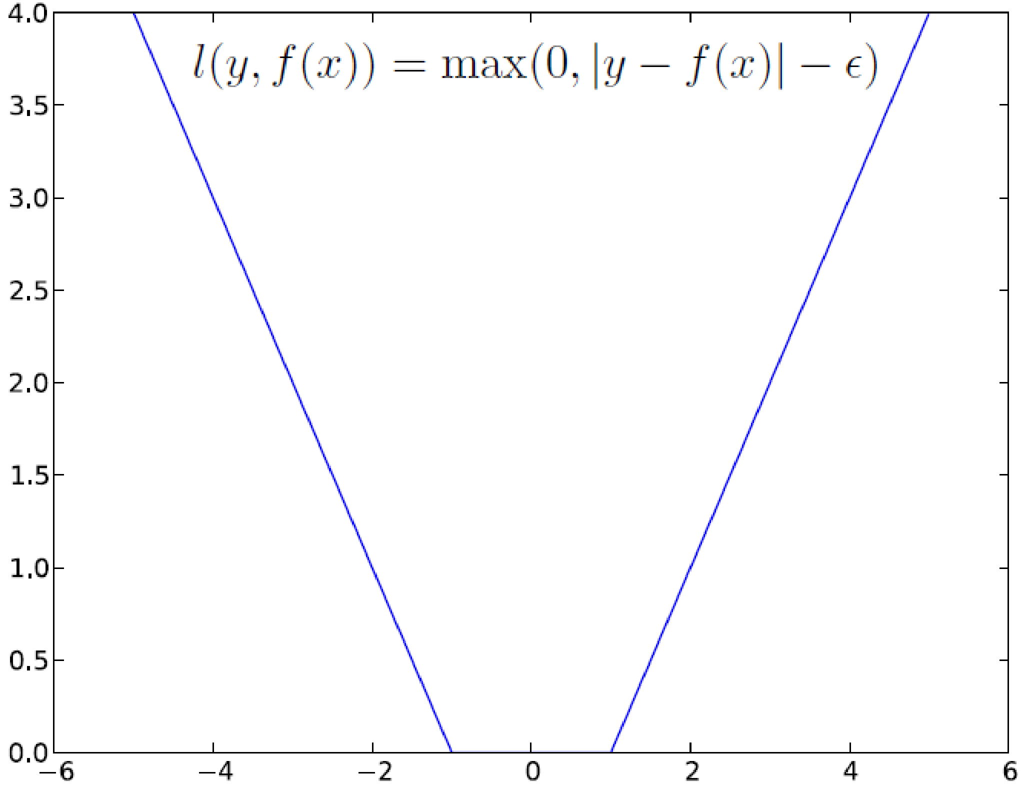


回归问题的一般框架：

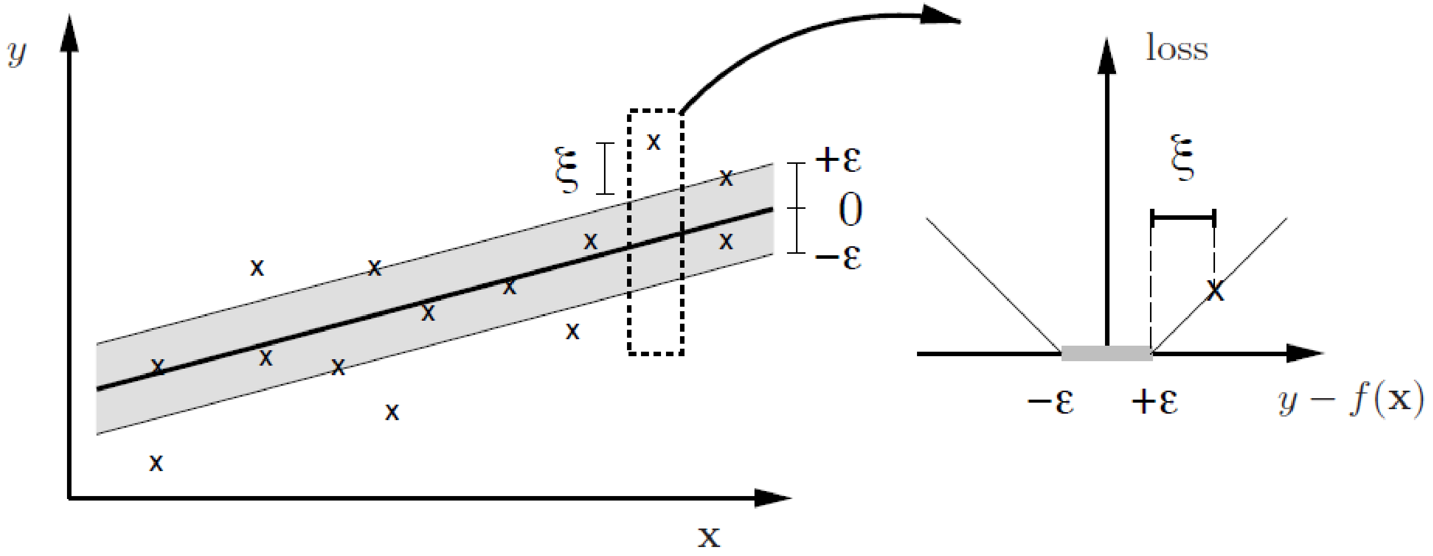
$$\min \quad \frac{\lambda}{2}|w|_2^2+\frac{C}{n}\sum_{i=1}^{n} l(\epsilon_i)$$

$$s.t \quad \epsilon_i = y_i - w^T x_i, \forall i \in [n]$$

- ϵ-insensitive损失函数：支持向量回归

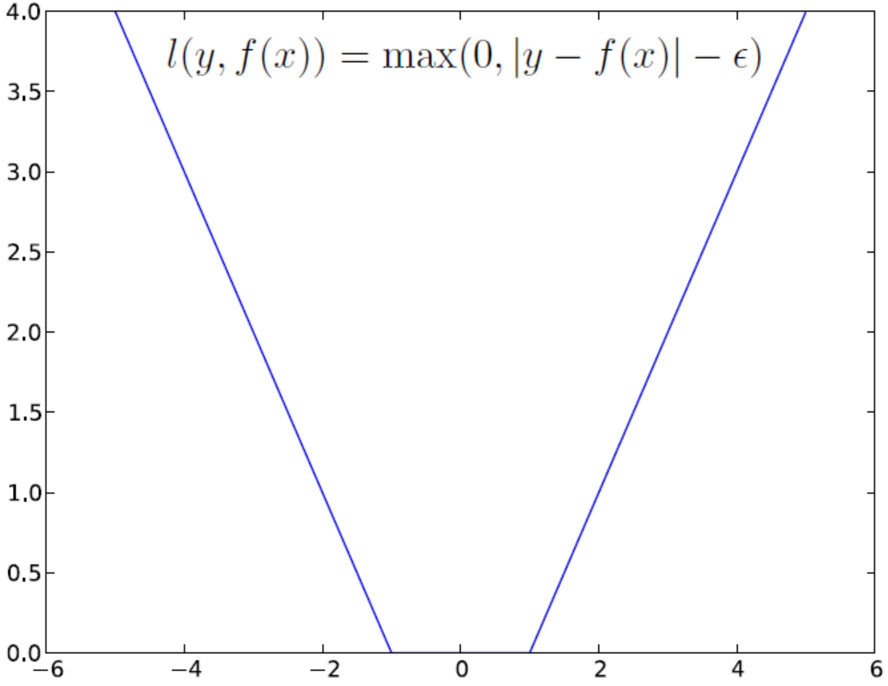


支持向量回归的性质：



- 鲁棒性好
- 对异常值(Outlier)不敏感
- 可以用与SVM同样的优化算法进行优化（多一倍的变量）

要点总结



4.1 回归问题的一般框架

4.2 支持向量回归对应的损失函数

4.3 支持向量回归的鲁棒性

THANK YOU！

Machine Learning Engineer
机器学习工程师微专业

稀牛学院 網易 NETEASE