# Machine Learning

2024 Fall, Final Review

**Xiangchen Tian**

Jan 2, 2025

清華大學
Tsinghua University

# Table of Contents

# Support Vector Machine(a.k.a. SVM)

- 支持向量机是一种监督学习算法
- 通过构造极大边距超平面以更好泛化
- 两种常见形式：Hard-SVM 和 Soft-SVM

# Hard-SVM

- 设训练数据集 $S=\{(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\cdots,(\mathbf{x}_m,y_m)\}$，其中 $\mathbf{x}_i\in\mathbb{R}^d$，$y_i\in\{-1,1\}$。
- 线性可分数据集：存在一个超平面 $(\mathbf{w},b)$ 使 $\forall i, y_i=\mathrm{sign}(\langle\mathbf{w},\mathbf{x}_i\rangle+b)$。
- Hard-SVM 只能处理线性可分数据集。

算法：

**Hard-SVM**

**input:** $(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_m,y_m)$

**solve:**

$$(\mathbf{w}_0,b_0)=\underset{(\mathbf{w},b)}{\operatorname{argmin}}\|\mathbf{w}\|^2 \text{ s.t. } \forall i,\ y_i(\langle\mathbf{w},\mathbf{x}_i\rangle+b)\geq 1 \quad (15.2)$$

**output:** $\hat{\mathbf{w}}=\frac{\mathbf{w}_0}{\|\mathbf{w}_0\|},\quad \hat{b}=\frac{b_0}{\|\mathbf{w}_0\|}$

分析：固定标度最大化边距，相当于固定边距最小化标度；最后归一化还原成固定标度最大化边距的结果。

# Soft-SVM

- Soft-SVM 可以处理非线性可分数据集。
- 思想：放宽限制为 $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geqslant 1 - \xi_i$

算法：

**Soft-SVM**

**input:** $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
**parameter:** $\lambda > 0$
**solve:**

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} \left( \lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right) \tag{15.4}$$

$$\text{s.t. } \forall i, \quad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0$$

**output:** $\mathbf{w}, b$

# hinge loss


- 定义 hinge loss $\ell^{\text{hinge}}(x) = \max\{0, 1-x\}$
- 定义 $L_S^{\text{hinge}}((\mathbf{w}, b)) = \frac{1}{m}\sum_{i=1}^{m} \ell^{\text{hinge}}(y_i \cdot (\langle \mathbf{w}, \mathbf{x}_i \rangle + b))$
- 注意到 (15.4) 式等价于

$$\min_{\mathbf{w}, b}\left(\lambda \|\mathbf{w}\|^2 + L_S^{\text{hinge}}((\mathbf{w}, b))\right)$$

  这就是标准的 regularized loss minimization problem，其中 $\lambda\|\mathbf{w}\|^2$ 是 $\ell_2$ 正则化。
- 因此 Soft-SVM 等价于一个 hinge loss、$\ell_2$ 正则化的优化问题。

- 以 Hard-SVM 为例（忽略 $b$），定义

$$g(\mathbf{w}) = \max_{\alpha \in \mathbb{R}^m, \alpha \geqslant 0} \sum_{i=1}^{m} \alpha_i \cdot (1 - y_i \cdot (\langle \mathbf{w}, \mathbf{x}_i \rangle)) = \begin{cases} 0 & \text{if } \forall i, y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geqslant 1 \\ +\infty & \text{otherwise} \end{cases}$$

则 (15.2) 式等价于

$$\begin{aligned} & \min_{\mathbf{w}} \left( \|\mathbf{w}\|^2 + g(\mathbf{w}) \right) \\ = & \min_{\mathbf{w}} \max_{\alpha \in \mathbb{R}^m, \alpha \geqslant 0} \left( \|\mathbf{w}\|^2 + \sum_{i=1}^{m} \alpha_i \cdot (1 - y_i \cdot (\langle \mathbf{w}, \mathbf{x}_i \rangle)) \right) = p^* \\ \geqslant & \max_{\alpha \in \mathbb{R}^m, \alpha \geqslant 0} \min_{\mathbf{w}} \left( \|\mathbf{w}\|^2 + \sum_{i=1}^{m} \alpha_i \cdot (1 - y_i \cdot (\langle \mathbf{w}, \mathbf{x}_i \rangle)) \right) = d^* \end{aligned} \tag{1}$$

- 在 Hard-SVM 中，由于特殊条件的满足，$p^* = d^*$，所以可以通过求解对偶问题来求解原问题。

- 对内部 $\mathbf{w}$ 取最小值，得到 $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$
- 带回对偶问题 (1) 式，得到

$$\max_{\alpha \in \mathbb{R}^m, \alpha \geqslant 0} \left( \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \right)$$

- 关键：只与 $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ 有关，而不与 $\mathbf{x}_i$ 有关，这是核技巧的基础。

- 思想：由于很多问题在原空间中线形不可分，希望先将原数据点映射到一个（更高维）空间中，然后在更高维空间中执行 SVM 算法。
- 算法：
  - 设原数据点定义域为 $X$, 映射函数为 $\phi : X \to F$，其中 $F$ 是特征空间（feature space）。
  - 给定原带标签数据集 $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$，构造像数据集 $\hat{S} = \{(\phi(\mathbf{x}_1), y_1), (\phi(\mathbf{x}_2), y_2), \cdots, (\phi(\mathbf{x}_m), y_m)\}$。
  - 在 $\hat{S}$ 上执行 SVM 算法，得到超平面 $(\mathbf{w}, b)$，对应一个线性分类器 $h : \psi(\mathbf{x}) \mapsto y$。
  - 预测原空间中 test set example $\mathbf{x}$ 为 $h(\psi(\mathbf{x}))$

# kernel

- kernel 是 feature space 中的 inner product。
- 给定 embedding $\phi : X \mapsto F$, 定义 kernel function

$$K(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

- $K$ 表征了 $\mathbf{x}, \mathbf{x}'$ 的相似度

- 很多 SVM 问题都可以总结为以下 general 问题的实例：

$$\min_{\mathbf{w}} \left(f(\langle \mathbf{w}, \mathbf{x}_1 \rangle, \langle \mathbf{w}, \mathbf{x}_2 \rangle, \cdots, \langle \mathbf{w}, \mathbf{x}_m \rangle) + R(\|\mathbf{w}\|_2)\right)$$

- Example 1: Soft-SVM, $R(a) = \lambda a^2$, $f(a_1, a_2, \cdots, a_m) = \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i a_i\}$
- Example 2: Hard-SVM, $R(a) = a^2$,

  $$f(a_1, a_2, \cdots, a_m) = \begin{cases} 0 & \text{if } \exists b \text{ s.t. } y_i(a_i + b) \geqslant 1 \\ +\infty & \text{otherwise} \end{cases}$$

- 而 $\mathbf{w} \in \text{span}(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m)$（reference, Theorem 16.1），因而事实上不会涉及 $\mathbf{x}_i$ 的具体值，只会涉及到 $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$。
- 因此，只需要知道 kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$，就能隐式在高维特征空间中执行 SVM 算法。

# characterizing kernel function

- 一个良定义的 kernel function 必须对应一个合理的 feature space embedding $\phi$。自然的问题：什么样的 kernel function 是合理的？

**Mercer's Theorem**

一个对称函数 $K : X \times X \to \mathbb{R}$ （对称：$K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}', \mathbf{x}), \forall \mathbf{x}, \mathbf{x}' \in X$） 可实现为某特征空间中的内积，当且仅当 $K$ 对应的 Gram matrix 是正定的。

- Gram matrix G : $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$.
- 证明思路：($\Rightarrow$)straight forward，($\Leftarrow$) 采用构造法，构造出一个特征空间和一个内积，使得这个内积对应的 kernel function 为 $K$。

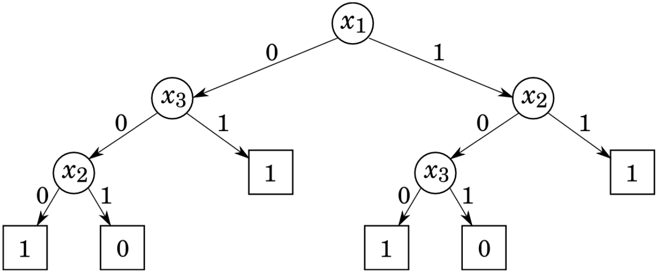
# Table of Contents

### Decision Tree

一个决策树是一个 Boolean Function $f : \mathbb{F}_2^n \to \mathbb{R}$ 的表示方法。它是一个含根二叉树，其中内部节点由某个 $i \in [n]$ 来标记，每个内部节点的出边被标记为 $0$ 或 $1$，每个叶子节点都有一个实数值，且要求没有 $i \in [n]$ 在一条从根到叶节点的路径上出现多于一次。

- 称决策树的叶节点总个数为决策树的大小 (size) $s$，称决策树根到叶节点路径的最大长度为决策树的深度 (depth) $d$。
- 根据定义，每个决策树都对应一个有 $n$ 个变量的布尔函数。

Example:




Note: 这棵树事实上对应函数 Sort$_3$, 其中 Sort$_3(x_1, x_2, x_3) = 1$ 当且仅当 $x_1 \geqslant x_2 \geqslant x_3$ 或 $x_1 \leqslant x_2 \leqslant x_3$。

**Theorem: Convert decision tree to low degree sparse function**

对任意有 $s$ 个叶节点的决策树 $T$，存在一个 degree 为 $\log(s/\epsilon)$，$L_0$-norm(sparsity) 为 $s^2/\epsilon$ 的布尔函数 $h$ 能够 $4\epsilon$-approximate $T$。

证明思路:

- 将决策树 $T$ 截断到深度为 $\log(s/\epsilon)$，最大误差为 $\epsilon$。
- 截断后的树 $T'$ 能用一个 $L_1(f) \leqslant s, \deg(f) = \log(s/\epsilon)$ 的布尔函数 $f$ 严格表示。
- 上述满足 $L_1(f) \leqslant s, \deg(f) = \log(s/\epsilon)$ 的布尔函数 $f$ 能用另一个满足 $L_0(h) \leqslant s^2/\epsilon, \deg(h) = \log(s/\epsilon)$ 的布尔函数 $h$ 来 $\epsilon$-approximate 表示。

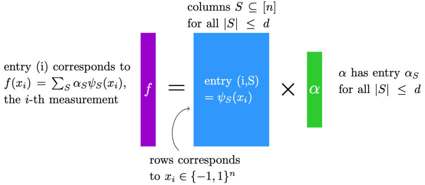
综上，$\|T - f\|^2 \leqslant \epsilon, \|f - h\|^2 \leqslant \epsilon$

$$\implies \|T - h\|^2 \leqslant 2\|T - f\|^2 + 2\|f - h\|^2 \leqslant 4\epsilon$$

# Practical Algorithms 1

由于使用上述定理提供的方法求 low degree approximate function $h$ 的复杂度较高，实际中采用以下几种算法：

- LMN: 设 $T$ 对应布尔函数 $f$，均匀采样 $m$ 个 $f$ 定义域内的点 $\{x_i, i \in [m]\}$，计算 $f(x_i)$，对每个 $|S| \leqslant \log(s/\epsilon)$ 的 $S$ 估计傅立叶系数 $\hat{f}(S) \approx \frac{1}{m}\sum_{i=1}^{m} f(x_i)\chi_S(x_i)$。最后返回
$$h = \sum_{S:|S|\leqslant \log(s/\epsilon)} \hat{f}(S)\chi_S$$
- Harmonica: 这个问题本质上是求一个在傅立叶基下稀疏的布尔函数，可以用 compress sensing 范式求解。



现实问题中的决策树：给定若干样例，希望从样例中学习决策树。

Example:

| Past trend | Open interest | Trading volume | Return |
|---|---|---|---|
| Positive | Low | High | Up |
| Negative | High | Low | Down |
| Positive | Low | High | Up |
| Positive | High | High | Up |
| Negative | Low | High | Down |
| Positive | Low | Low | Down |
| Negative | High | High | Down |
| Negative | Low | High | Down |
| Positive | Low | Low | Down |
| Positive | High | High | Up |

与上面的 approach 不同，另一种方法是递归选取最重要的变量作为内部节点划分样例。

算法：

**function** LEARN-DECISION-TREE(*examples, attributes, parent examples*) **returns** 一棵树

  **if** *examples*不为空 **then return** PLURALITY-VALUE(*parent examples*)
  **else if** 所有*examples*有相同的分类 **then return** 分类
  **else if** *attributes*为空 **then return** PLURALITY-VALUE(*examples*)
  **else**
    $A \leftarrow \mathrm{argmax}_{a \in attributes}$ IMPORTANCE($a$, *examples*)
    *tree* ← 一个以测试$A$为根的新的决策树
    **for each** $A$中的值$v$ **do**
      $exs \leftarrow \{e : e \in examples$ **and** $e.A = v\}$
      *subtree*←LEARN-DECISION-TREE (*exs, attributes*−*A, examples*)
      将一个带有标签（$A = v$）和子树 *subtree*的分支加入*tree*
   **return** *tree*

解释：每次根据某种准则选择最重要的变量 $i$（若某变量的取值几乎决定了最终的类别是什么，说明这个变量较重要）

- Gini Index 是一种判断某个变量是否重要的一种准则。
- 定义：对变量 A，$\mathrm{Gini}(A) = \sum_a p(A = a)\mathrm{Gini}(a)$, 其中 $\mathrm{Gini}(a) = 1 - \sum_i p_i^2$（详见课件例子）
- $\mathrm{Gini}(a)$ 越小表示区分度越好（如果一个变量 $A$ 取正时所有结果都是正，一个变量取负时所有结果都是负，则 $\mathrm{Gini}(A) = 0$），故每次划分变量时选择 $\mathrm{Gini}(A)$ 最小的变量 $A$。
- 其他准则：Information Gain

# Table of Contents

- 思想：希望能有一种方法能够聚合多个弱学习器（每个弱学习器可以只比随机猜测表现好一点），使得整体学习器的性能更好。
- 一次性学习强学习器可能计算复杂度上承担不起，但每个弱学习器的计算复杂度较低，聚合多个弱学习器的计算复杂度可以接受。

# AdaBoost (a.k.a Adaptive Boosting)

- AdaBoost 是最典型的 boosting 算法。
- AdaBoost 思想：在简单假设类上构建线性预测期作为更强大的假设类。

算法:

`AdaBoost`

**input:**
training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
weak learner WL
number of rounds $T$
**initialize** $\mathbf{D}^{(1)} = (\frac{1}{m}, \ldots, \frac{1}{m})$.
**for** $t = 1, \ldots, T$:
invoke weak learner $h_t = \mathrm{WL}(\mathbf{D}^{(t)}, S)$
compute $\epsilon_t = \sum_{i=1}^{m} D_i^{(t)} \mathbb{1}_{[y_i \neq h_t(\mathbf{x}_i)]}$
let $w_t = \frac{1}{2} \log\left(\frac{1}{\epsilon_t} - 1\right)$
update $D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\mathbf{x}_i))}{\sum_{j=1}^{m} D_j^{(t)} \exp(-w_t y_j h_t(\mathbf{x}_j))}$ for all $i = 1, \ldots, m$
**output** the hypothesis $h_s(\mathbf{x}) = \mathrm{sign}\left(\sum_{t=1}^{T} w_t h_t(\mathbf{x})\right)$.

**Theorem: AdaBoost Training Error Upper Bound**

$S$ 是一个训练集，假设在 AdaBoost 的每一次迭代中，弱分类器返回的假设满足 $\epsilon_t \leq \frac{1}{2} - \gamma$。则 AdaBoost 输出的最终假设的训练误差满足以下不等式：

$$L_S(h_s) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}[h_s(x_i) \neq y_i] \leq \exp(-2\gamma^2 T)$$

证明：对于每一轮 $t$，记 $f_t = \sum_{p<t} w_p h_p$，因此 AdaBoost 的输出为 $f_T$。此外，记

$$Z_t = \sum_{i=1}^{m} \exp(-y_i f_t(x_i))$$

# AdaBoost Analysis

注意到对于任何假设都有 $\mathbb{1}[h(x) \neq y] \leq \exp(-yh(x))$。因此，训练误差满足：

$$L_S(f_T) \leq Z_T$$

所以我们只需证明 $Z_T \leq \exp(-2\gamma^2 T)$。为了 bound 住 $Z_T$，我们将其重写为：

$$Z_T = \frac{Z_T}{Z_{T-1}} \cdot \frac{Z_{T-1}}{Z_{T-2}} \cdots \frac{Z_2}{Z_1} \cdot \frac{Z_1}{Z_0}$$

其中我们使用了 $Z_0 = 1$，因为 $f_0 \equiv 0$。现在我们只需证明对于每一轮 $t$:

$$\frac{Z_{t+1}}{Z_t} \leq \exp(-2\gamma^2)$$

为了证明上式，首先通过简单的归纳推理可知对于所有的 $t$ 和 $i$ 都有：

$$D_i^{(t+1)} = \frac{\exp(-y_i f_t(x_i))}{\sum_{j=1}^m \exp(-y_j f_t(x_j))}$$

因此:

$$\frac{Z_{t+1}}{Z_t} = \frac{\sum_{i=1}^m \exp(-y_i f_{t+1}(x_i))}{\sum_{j=1}^m \exp(-y_j f_t(x_j))} = \frac{\sum_{i=1}^m \exp(-y_i f_t(x_i)) \exp(-y_i w_{t+1} h_{t+1}(x_i))}{\sum_{j=1}^m \exp(-y_j f_t(x_j))}$$

$$\Longrightarrow \frac{Z_{t+1}}{Z_t} = \exp(-w_{t+1})\,(1-\epsilon_{t+1}) + \exp(w_{t+1})\epsilon_{t+1} = 2\sqrt{\epsilon_{t+1}(1-\epsilon_{t+1})}$$

根据我们的假设，$\epsilon_{t+1} \leq \frac{1}{2} - \gamma$。由于函数 $g(a) = a(1-a)$ 在区间 $[0, \frac{1}{2}]$ 上是单调递增的，我们得到:

$$2\sqrt{\epsilon_{t+1}(1-\epsilon_{t+1})} \leqslant 2\sqrt{\left(\frac{1}{2}-\gamma\right)\left(\frac{1}{2}+\gamma\right)} \leqslant \sqrt{1-4\gamma^2}$$

因此，$\frac{Z_{t+1}}{Z_t} \leq \exp(-2\gamma^2)$。 $\square$

# Table of Contents

4 PCA

# Principal Component Analysis (a.k.a. PCA)



- PCA 是一种降维技术，将高维空间中的数据映射到低维空间。
- 详细请参考计算机与人工智能应用数学
- power method：一种高效求解最大本征值和对应的本征向量的方法

# Table of Contents

5 Nearest Neighbor and LSH

# Nearest Neighbor

- Nearest Neighbor 是一种非参数化模型。（数据集就是参数）
- 直接用遍历所有点的方式找到邻居的方法非常耗时，希望有一种数据结构能够高效返回近似最近邻
- LSH algorithm

# LSH

# Table of Contents

# Adding images

# Table of Contents

7 Summary

# Machine Learning

*Thank you for listening!*
*Any questions?*