支持向量机算法

陈 鑫

2023 年 4 月 7 日

目录

目录

支持向量机（Support Vector Machines，SVM）是一种二分类模型，它的思想源于解析几何，其基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM 对于线性不可分的样本，可以通过灵活多变的核方法，将数据投影到高维空间，利用高维空间的超平面来分离数据，再将已分离的数据重新投影回原空间，得到原空间中正负样本的一个非线性边界，这使它成为实质上的非线性分类器。SVM 的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM 的学习算法就是求解凸二次规划的最优化算法。

支持向量机是应用于二元分类问题中的一种监督学习方法。在处理二元分类问题时，要寻找一个将两类事物相分离的超平面，在二维空间时，就是一条直线将两类事物区分开来。通常这样的超平面会有很多个。支持向量机算法的目标是构造能正确划分训练数据集并且与要分离的两类采样有最大几何间隔的分离超平面。具有这种特征的分离超平面不仅能完美地区分训练数据，而且对测试数据也有较好的分类预测能力。

定义

（数据集的线性可分性）给定一个数据集 $S = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$，其中 $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$，$y^{(i)} \in \{-1, +1\}, i = 1, 2, \cdots, m$。如果存在某个超平面 $\boldsymbol{S}$：$\boldsymbol{x}^T\boldsymbol{w} + b = 0$ 能够将数据集的正样本点和负样本点完全正确地划分到超平面的两侧，即对所有的 $y^{(i)} = +1$ 的实例 i，有 $\boldsymbol{x}^T\boldsymbol{w} + b \geq 0$，对所有 $y^{(i)} = -1$ 的实例 i，有 $\boldsymbol{x}^T\boldsymbol{w} + b < 0$，则称数据集 S 为线性可分数据集；否则，称数据集 S 线性不可分。

感知机

感知机 (perceptron) 是一种二元分类的线性分类模型，其输入为实例的特征向量，输出为实例的类别 $(+1,-1)$。感知机学习旨在求出将训练数据进行线性划分的分离超平面，为此，导入基于误分类的损失函数，利用梯度下降法对损失函数进行极小化，求得感知机模型，具有算法简单，易于实现的特点。感知机预测是用学习得到的感知机模型对新的输入实例进行分类。它是神经网络与支持向量机的基础。

定义

感知机：取定特征 $\boldsymbol{x} \in \mathbb{R}^n$，标签 $y \in \{-1,+1\}$，称 $f(\boldsymbol{x}) = sign(\boldsymbol{x}^T\boldsymbol{w} + b)$ 为感知机。其中 $\boldsymbol{w}$ 和 b 为感知机模型参数，$\boldsymbol{w} \in \mathbb{R}^n$ 称为权值 *(weight)* 或权值向量 *(weight vector)*，b 称为偏置 *(bias)*，$\boldsymbol{x}^T\boldsymbol{w}$ 表示 $\boldsymbol{x}$ 和 $\boldsymbol{w}$ 的内积。$sign$ 是符号函数

$$sign(\boldsymbol{x}) = \begin{cases} +1, \boldsymbol{x} \geq 0, \\ -1, \boldsymbol{x} < 0 \end{cases}$$

感知机

对于感知机而言，损失函数为误分类点到超平面的总距离。可知损失函数越小，说明被误分类的点到超平面的距离较近，且被误分类点较少，如果损失函数为 0，说明没有误分类点。

由空间解析几何知道，空间中任一点 $\boldsymbol{x} \in \mathbb{R}^n$ 到超平面 L 距离 d：

$$d(\boldsymbol{x}, L) = \frac{|\boldsymbol{x}^T\boldsymbol{w} + b|}{||\boldsymbol{w}||}$$

其中，$||\boldsymbol{w}||$ 是 $\boldsymbol{w}$ 的 L_2 范数。对于误分类的点 $(\boldsymbol{x}^{(i)}, y^{(i)})$ 来说，有 $-y^{(i)}(\boldsymbol{x}^{(i)T}\boldsymbol{w} + b) > 0$。因为当 $\boldsymbol{x}^{(i)T}\boldsymbol{w} + b > 0$，$y = -1$，当 $\boldsymbol{x}^{(i)}\boldsymbol{w} + b < 0$，$y = +1$。因此，误分类点 $\boldsymbol{x}^{(i)}$ 到超平面 L 的距离是 $-y^{(i)}\frac{\boldsymbol{x}^{(i)T}\boldsymbol{w}+b}{||\boldsymbol{w}||}$。正确分类的点无损失，所有的误分类点 M 到超平面的总距离为：$-\frac{1}{||\boldsymbol{w}||}\sum_{\boldsymbol{x}^{(i)} \in M} y^{(i)}(\boldsymbol{x}^{(i)T}\boldsymbol{w} + b)$，因要求的是损失函数的最小值，可以忽略总距离的常系数 $||\boldsymbol{w}||$。可得其损失函数为：

$$\ell(\boldsymbol{w}, b) = -\sum_{\boldsymbol{x}^{(i)} \in M} y^{(i)}(\boldsymbol{x}^{(i)T}\boldsymbol{w} + b)$$

这就是感知机的经验损失函数。

感知机算法描述

由感知机的经验损失函数，得到其对 $\boldsymbol{w}$ 和 b 的梯度分别为：

$$\nabla_{\boldsymbol{w}}\ell(\boldsymbol{w},b) = -\sum_{\boldsymbol{x}^{(i)}\in M} y^{(i)}\boldsymbol{x}^{(i)}$$

$$\nabla_{b}\ell(\boldsymbol{w},b) = -\sum_{\boldsymbol{x}^{(i)}\in M} y^{(i)}$$

因此，由梯度下降算法求其经验损失最小化，得到如下算法。

Algorithm 1: 感知机算法

Input: $S = \{(\boldsymbol{x}^{(1)},y^{(1)}),(\boldsymbol{x}^{(2)},y^{(2)}),\cdots,(\boldsymbol{x}^{(m)},y^{(m)})\}$

Output: $\boldsymbol{w}, b$

初始化 $\boldsymbol{w}$，b 和 $done = False$

while *not done* **do**

  $done = True$

  foreach $i = 1,\cdots,m$ **do**

    if $y^{(i)}sign(\boldsymbol{x}^{(i)T}\boldsymbol{w}+b) \le 0$ **then**

      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y^{(i)}\boldsymbol{x}^{(i)}$

      $b \leftarrow b + \eta y^{(i)}$，其中 $0 \le \eta \le 1$，称为学习率

      $done = False$

由 Python 实现该算法的代码 perceptron.py：

```
import numpy as np
class Perceptron:
    def __init__(self):
        self.w = None
        self.b = None
    def fit(self, X, y, eta=0.1): # eta为学习率
        m, n = X.shape # m样本数,n每个样本的特征数
        w = np.zeros((n, 1)) # 初始化参数w为全0的矩阵
        b = 0
        done = False # done为标识是否已经完成无误分类的训练
        while not done:
            done = True
            for i in range(m):
                x = X[i].reshape(1, -1) # 转化为合适的形状，以便下面计算
                if y[i] * (x.dot(w) + b) <= 0: # 判断是否是误分类点
                    w = w + eta * y[i] * x.T # 使用梯度下降算法迭代
                    b = b + eta * y[i]
                    done = False
        self.w = w # 得到无误分类点时的参数
        self.b = b
    def predict(self, X): # 对新样本进行类别的预测
        return np.sign(X.dot(self.w) + self.b)
```

目录

支持向量机的基本概念

支持向量机（Support Vector Machines，SVM）是一种二分类模型，它的思想源于解析几何，其基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM 对于线性不可分的样本，可以通过灵活多变的核方法，将数据投影到高维空间，利用高维空间的超平面来分离数据，再将已分离的数据重新投影回原空间，得到原空间中正负样本的一个非线性边界，这使它成为实质上的非线性分类器。SVM 的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM 的学习算法就是求解凸二次规划的最优化算法。

定义

（数据集的线性可分性）给定一个数据集 $S=\{(\boldsymbol{x}^{(1)},y^{(1)}),(\boldsymbol{x}^{(2)},y^{(2)}),\cdots,(\boldsymbol{x}^{(m)},y^{(m)})\}$，其中 $\boldsymbol{x}^{(i)}\in\mathbb{R}^n$，$y^{(i)}\in\{-1,+1\},i=1,2,\cdots,m$。如果存在某个超平面 S：$\boldsymbol{x}^T\boldsymbol{w}+b=0$ 能够将数据集的正样本点和负样本点完全正确地划分到超平面的两侧，即对所有的 $y^{(i)}=+1$ 的实例 i，有 $\boldsymbol{x}^T\boldsymbol{w}+b\geq 0$，对所有 $y^{(i)}=-1$ 的实例 i，有 $\boldsymbol{x}^T\boldsymbol{w}+b<0$，则称数据集 S 为线性可分数据集；否则，称数据集 S 线性不可分。

支持向量机的目标

感知机的目标是将训练集分开，只要是能将样本分开的超平面都满足要求，而这样的超平面有很多。支持向量机本质上和感知机类似，要求却更加苛刻，支持向量机算法是有选择地计算出一条拓展性最强的分离直线，因此对测试数据也有较好的分类预测能力。支持向量机算法的核心思想是计算一条最为中立的分离直线，该直线既不偏向训练数据中的正样本，也不偏向负样本，其中立性通过间隔概念来体现的。因为在分类过程中，那些远离超平面的点是安全的，而那些容易被误分类的点是离超平面很近的点，支持向量机就是要重点关注这些离超平面很近的点，在分类正确的同时，让离超平面最近的点到超平面的间隔最大。支持向量机算法的目标就是求解数据样本到分离直线的最小距离的最大值，也就是间隔的最大化

$$\max_{\boldsymbol{w}\in\mathbb{R}^n,b\in\mathbb{R}} \delta_S(\boldsymbol{w},b) = \max_{\boldsymbol{w}\in\mathbb{R}^n,b\in\mathbb{R}} \{\min_{1\le i\le m} d(\boldsymbol{x}^{(i)},L)\} \tag{1}$$

在训练数据可分离的前提下，式(1)的最优解就是间隔最大的分离直线。

经过一系列变换，支持向量机算法的优化目标由式(1)的间隔最大化问题转化为与之等价的式

$$\begin{aligned} &\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 \\ &\text{约束：} y^{(i)}(\boldsymbol{x}^{(i)T}\boldsymbol{w}+b) \ge 1, i=1,2,\cdots,m \end{aligned} \tag{2}$$

这是一个标准的凸优化问题。

支持向量机的对偶

支持向量机算法从其原始定义的无约束优化问题(1)转化为带约束的凸优化问题(2)，而凸优化问题中的对偶理论则是用拉格朗日乘子将原始问题的带约束条件转至目标函数。因此，可将式(2)转化为不带约束条件的

$$L(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}||\boldsymbol{w}||^2 + \sum_{i=1}^{m} \lambda_i (1 - y^{(i)}(\boldsymbol{x}^{(i)T}\boldsymbol{w} + b)) \tag{3}$$

其中，λ_i 是约束条件 $y^{(i)}(\boldsymbol{x}^{(i)T}\boldsymbol{w} + b) \geq 1$ 对应的拉格朗日乘子。设 $G(\boldsymbol{\lambda})$ 是(2)的对偶函数，即 $G(\boldsymbol{\lambda}) = \min_{\boldsymbol{w},b} L(\boldsymbol{w}, b, \boldsymbol{\lambda})$。要计算 $G(\boldsymbol{\lambda})$ 的具体表达式，将(3)重新组合如下：

$$L(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{m} \lambda_i y^{(i)} \boldsymbol{x}^{(i)T}\boldsymbol{w} - \left(\sum_{i=1}^{m} \lambda_i y^{(i)} \right) b + \sum_{i=1}^{m} \lambda_i \tag{4}$$

式(3)表达的是一个无约束凸优化问题。它的最优解 $\boldsymbol{w}^*$、b^* 必须满足

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}^*, b^*, \boldsymbol{\lambda}) = \boldsymbol{w}^* - \sum_{i=1}^{m} \lambda_i y^{(i)} \boldsymbol{x}^{(i)} = \boldsymbol{0} \tag{5}$$

$$\nabla_b L(\boldsymbol{w}^*, b^*, \boldsymbol{\lambda}) = \sum_{i=1}^{m} \lambda_i y^{(i)} = 0 \tag{6}$$

仅当式(6)成立时，$G(\boldsymbol{\lambda})$ 的值才是有限的。事实上，式(4)中 b 的系数恰为 $\sum_{i=1}^{m}\lambda_i y^{(i)}$。若 $\sum_{i=1}^{m}\lambda_i y^{(i)}\neq 0$，不妨设 $\sum_{i=1}^{m}\lambda_i y^{(i)}>0$，则 $\lim\limits_{b\to+\infty}L(\boldsymbol{w},b,\boldsymbol{\lambda})=-\infty$。可见，此时 $G(\boldsymbol{\lambda})=-\infty$。假设式(6)成立。此时 $G(\boldsymbol{\lambda})=L(\boldsymbol{w}^*,b^*,\boldsymbol{\lambda})$。从式(5)可知，$\boldsymbol{w}^*=\sum\limits_{i=1}^{m}\lambda_i y^{(i)}\boldsymbol{x}^{(i)}$。将此式代入式(4)，并结合式(6)得到

$$G(\boldsymbol{\lambda})=\begin{cases}-\infty, & \text{如果}\sum\limits_{i=1}^{m}\lambda_i y^{(i)}\neq 0\\ \sum\limits_{i=1}^{m}-\frac{1}{2}\sum\limits_{i,j=1}^{m}\lambda_i\lambda_j y^{(i)}y^{(j)}x^{(i)T}x^{(j)}, & \text{如果}\sum\limits_{i=1}^{m}\lambda_i y^{(i)}=0\end{cases}\tag{7}$$

式(2)中凸优化问题的对偶问题是

$$\max_{\boldsymbol{\lambda}>0}G(\boldsymbol{\lambda})\tag{8}$$

由此可见，在式(8)表示的对偶问题中，不必考虑那些使得 $\sum\limits_{i=1}^{m}\lambda_i y^{(i)}\neq 0$ 的拉格朗日乘子 $\boldsymbol{\lambda}$。事实上，这样的乘子的对偶函数 $G(\boldsymbol{\lambda})=-\infty$，从而它不可能是式(8)的最优解。因此，可将对偶问题写成为

$$\begin{aligned}&\max G(\boldsymbol{\lambda})=\sum_{i=1}^{m}\lambda_i-\frac{1}{2}\sum_{i,j=1}^{m}\lambda_i\lambda_j y^{(i)}y^{(j)}\boldsymbol{x}^{(i)T}\boldsymbol{x}^{(j)}\\ &\text{约束：}\sum_{i=1}^{m}\lambda_i y^{(i)}=0,\lambda_i\geq 0,i=1,2,\cdots,m\end{aligned}\tag{9}$$

支持向量机优化算法

SMO (Sequential Minimal Optimization) 序列最小优化算法是求解式(9)的一个高效优化算法。SMO 算法是 4.5 节中介绍的坐标下降算法在支持向量机对偶问题中的具体体现。SMO 算法的核心思想是每一次选取 2 个变量进行调整。在每一轮搜索中选取两个变量 λ_i 和 λ_j，并固定其他变量的取值。SMO 算法所需的一些中间结果：

$$K_{i,j} = \boldsymbol{x}^{(i)T}\boldsymbol{x}^{(j)}$$

$$E_i = \sum_{t=1}^{m} \lambda_t^* y^{(t)} K_{t,i} - y^{(i)}$$

$$L_{i,j} = \begin{cases} 0, & y^{(i)} = y^{(j)} \\ \max\{0, \lambda_j^* - \lambda_i^*\}, & y^{(i)} \neq y^{(j)} \end{cases}$$

$$H_{i,j} = \begin{cases} \lambda_i^* + \lambda_j^*, & y^{(i)} = y^{(j)} \\ +\infty, & y^{(i)} \neq y^{(j)} \end{cases}$$

SMO 算法描述

Algorithm 2: SMO 算法

Input: m 个训练数据 $S = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(m)}, y^{(m)})\}, \boldsymbol{x} \in \mathbb{R}^n, y \in \mathbb{R}$

Output: $h_{\boldsymbol{w}^*, b^*}$

$\boldsymbol{\lambda} = \mathbf{0}, b = 0$

foreach i, j: **do**

  $K_{i,j} = \boldsymbol{x}^{(i)T}\boldsymbol{x}^{(j)}$

for $r = 1, 2, \cdots, N$: **do**

  for $i = 1, 2, \cdots, m$: **do**

    for $j = 1, 2, \cdots, m$: **do**

      $\delta_j = \max\{L_{i,j}, \min\{\lambda_j + \frac{E_j - E_i}{2K_{i,j} - K_{i,i} - K_{j,j}}, H_{i,j}\}\} - \lambda_j$

      $\lambda_j \leftarrow \lambda_j + \delta_j$

      $\lambda_i \leftarrow \lambda_i - y^{(i)} y^{(j)} \delta_j$

      if $\lambda_i > 0$: **then**

        $b = y^{(i)} - \sum_{t=1}^{m} \lambda_t y^{(t)} K_{t,i}$

      else if $\lambda_j > 0$: **then**

        $b = y^{(j)} - \sum_{t=1}^{m} \lambda_t y^{(t)} K_{t,j}$

$\boldsymbol{w} = \sum_{i=1}^{m} \lambda_i y^{(i)} \boldsymbol{x}^{(i)}$

return $h(\boldsymbol{x}) = Sign(\boldsymbol{x}^T \boldsymbol{w} + b)$

由 Python 实现该算法的代码 svm_smo.py

```
import numpy as np
class SVM:
    def __init__(self):
        self.Lambda = None
        self.w,self.b = None, None
    def get_H(self, Lambda, i, j, y):
        if y[i] == y[j]:
            return Lambda[i] + Lambda[j]
        else:
            return float("inf")
    def get_L(self, Lambda, i, j, y):
        if y[i] == y[j]:
            return 0.0
        else:
            return max(0, Lambda[j] - Lambda[i])
    def smo(self, X, y, K, N):
        m, n = X.shape
        Lambda = np.zeros((m, 1))
        epsilon = 1e-6
        for t in range(N):
            for i in range(m):
                for j in range(m):
                    D_ij = 2 * K[i][j] - K[i][i] - K[j][j]
                    if abs(D_ij) < epsilon:
                        continue
                    E_i = K[:, i].dot(Lambda * y) - y[i]
```

```
                E_j = K[:, j].dot(Lambda * y) - y[j]
                delta_j = 1.0 * y[j] * (E_j - E_i) / D_ij
                H_ij = self.get_H(Lambda, i, j, y)
                L_ij = self.get_L(Lambda, i, j, y)
                if Lambda[j] + delta_j > H_ij:
                    delta_j = H_ij - Lambda[j]
                    Lambda[j] = H_ij
                elif Lambda[j] + delta_j < L_ij:
                    delta_j = L_ij - Lambda[j]
                    Lambda[j] = L_ij
                else:
                    Lambda[j] += delta_j
                delta_i = - y[i] * y[j] * delta_j
                Lambda[i] += delta_i
                if Lambda[i] > epsilon:
                    b = y[i] - K[:, i].dot(Lambda * y)
                elif Lambda[j] > epsilon:
                    b = y[j] - K[:, j].dot(Lambda * y)
    self.Lambda = Lambda
    self.b = b
def fit(self, X, y, N=10):
    K = X.dot(X.T)
    self.smo(X, y, K, N)
    self.w = X.T.dot(self.Lambda * y)
    return self.w, self.b
def predict(self, X):
    return np.sign(X.dot(self.w) + self.b)
```

案例实战 6.2：使用支持向量机 SMO 算法对山鸢尾花和非山鸢尾花进行分类预测。实现 SMO 算法的文件 svm_smo.py，进行分类的文件 iris_svm.py

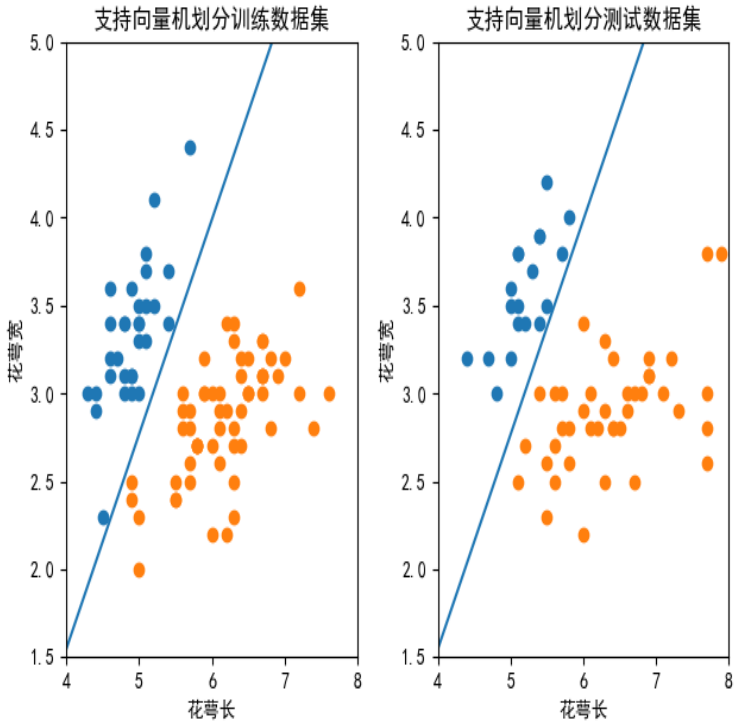


图 1:（山）鸢尾花的分类

目录

核方法

前面的讨论是假定了训练数据中的正样本和负样本是存在分离超平面的，但实际情况是，训练数据的正负样本之间可能不存在分离超平面。这时，前面的算法就没有可行解，因此前两节的算法就无效了，从而限制了支持向量机算法的应用。可通过核方法和软间隔支持向量机算法对原有算法进行改进，从而拓展了支持向量机的应用场景。

核方法的思想是，相同的数据在低维空间中线性不可分，但其在高维的空间中可能是线性可分的。因此，如果正负样本之间在原来的空间中不存在分离超平面，可以将它们以某种形式投影到高维空间中，此时，可以使用支持向量机算法计算数据在高维空间中的分离超平面，然后再将计算所得的高维空间的分离超平面重新投影回原来的（低维）空间，从而得到原空间中正负样本之间的一个非线性边界。

案例实战 6.3：在一维空间（x 轴）中，有正样本 $(-4,-3,-2,4,5)$ 和负样本 $(-1,0,1,2,3)$，请判断它们在一维空间中是否线性可分？如果将数据投影到二维空间，能否让其变成线性可分？请绘制图像说明。linear_unseparable.py

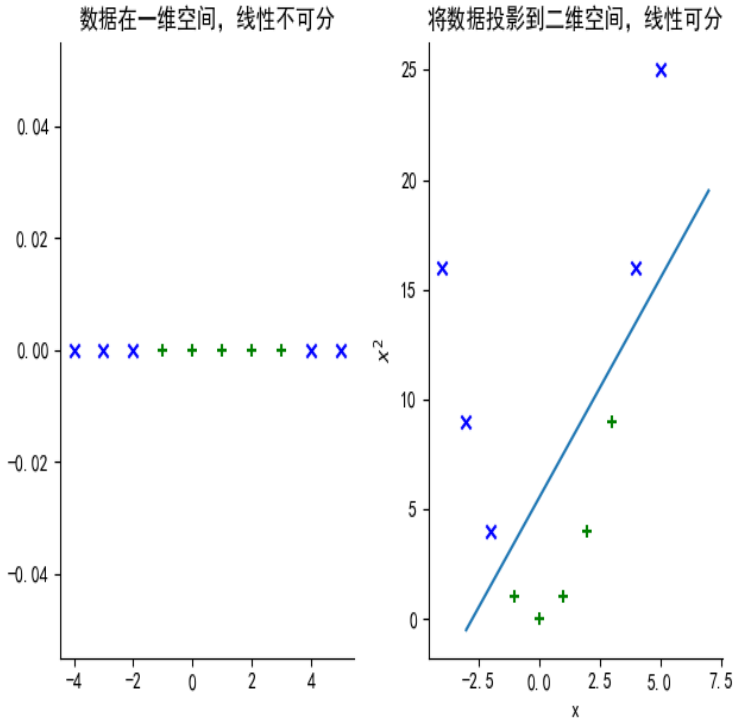
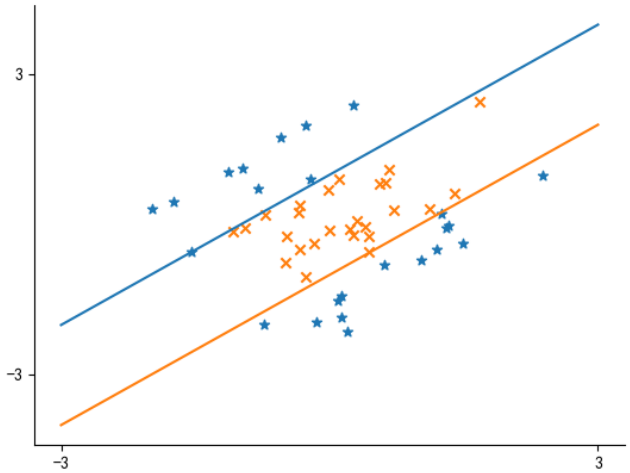
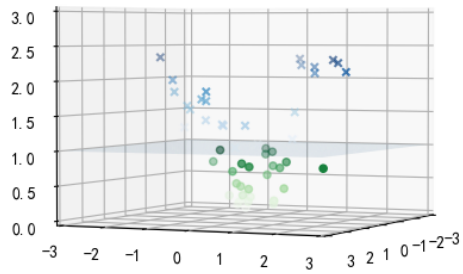


图 2: 数据从低维空间投影到高维空间

案例实战 6.4 在二维空间中，也有很多数据是线性不可分的。例如，随机产生一组数据，分别表示零件的长度和宽度（赋以一定的意义），当长宽的差绝对值小于 1 时，表示产品合格，为正品。当长宽的差绝对值大于 1 时，表示产品不合格，为次品。
low_dim_project_to_high_dim.py



(a) 2 维数据



(b) 投影至 3 维空间

图 3: 正负样本在二维空间以及将其投影到三维空间

将在低维空间中不可线性分离的数据投影到高维空间，有可能变成可以线性分离。一般情况下，将低维空间的数据投影到高维空间，需要占用大量的存储空间，而且其计算时间复杂度也会大幅度增加。

核方法是一类把低维空间的非线性可分问题，转化为高维空间的线性可分问题的方法。核方法不仅仅用于 SVM，还可以用于其他数据为非线性可分的算法。核方法的理论基础是 Cover's theorem，指的是对于非线性可分的训练集，可以大概率通过将其非线性映射到一个高维空间来转化为线性可分的训练集。

核方法关注的对象是一类特殊的高维投影，它能使支持向量机算法在高维空间中依然有较低的时间和空间复杂度。这要求投影 ϕ 必须满足：对于任意的两个 n 维向量 $\boldsymbol{x}$ 和 $\boldsymbol{z}$，它们投影的内积 $\phi(\boldsymbol{x})^T\phi(\boldsymbol{z})$ 具有高效的计算方法。

定义

（核函数）设 ϕ：$\mathbb{R}^n \to \mathbb{R}^N$ 为 n 维空间到 N 维空间的投影，对任意的 $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^n$，定义投影 ϕ 的核函数为 $K_\phi(\boldsymbol{x}, \boldsymbol{z}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{z})$，即核函数输入两个向量，它返回的值与两个向量分别作 ϕ 投影，然后做内积的结果相同。

核技巧是一种利用核函数直接计算 $\phi(\boldsymbol{x})^T \phi(\boldsymbol{z})$，以避开分别计算 $\phi(\boldsymbol{x})$ 和 $\phi(\boldsymbol{z})$，从而加速核方法计算的技巧。

例如多项式投影 $\phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$，对于任意 $\boldsymbol{x} = (x_1, x_2)$ 和 $\boldsymbol{z} = (z_1, z_2)$，有 $\phi(\boldsymbol{x})^T \phi(\boldsymbol{z}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)^T (z_1^2, z_2^2, \sqrt{2}z_1z_2) = x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 = (x_1z_1 + x_2z_2)^2 = (\boldsymbol{x}^T\boldsymbol{z})^2$。由此可见，为了计算 $\phi(\boldsymbol{x})^T \phi(\boldsymbol{z})$ 只需计算 $\boldsymbol{x}^T\boldsymbol{z}$ 就可以了，不必关注 $\phi(\boldsymbol{x})$ 和 $\phi(\boldsymbol{z})$ 的取值，从而大大降低了计算的复杂度。（如果换个系数 $\sqrt{3}$ 计算量就大多了）。

核方法的中心思想

SMO 算法只涉及计算特征的内积 $K_{i,j} = \boldsymbol{x}^{(i)T}\boldsymbol{x}^{(j)}$，并不涉及特征本身，算法返回的模型 $h(\boldsymbol{x}) = sign(\sum_{i=1}^{m} \lambda_i^* y^{(i)}\boldsymbol{x}^{(i)T}\boldsymbol{x} + b^*)$，也只与内积 $\boldsymbol{x}^{(i)T}\boldsymbol{x}$ 有关，而不涉及特征本身。因此，即使投影变换 ϕ 将特征转化为高维特征，支持向量机也不需要具体使用 $\phi(\boldsymbol{x})$ 的信息，而只需要通过内积计算就可以完成模型的训练和预测。只要对任意向量 $\boldsymbol{x}$ 和 $\boldsymbol{z}$，能够高效计算其投影内积 $\phi(\boldsymbol{x})^T\phi(\boldsymbol{z})$，算法就能有较低的时间和空间复杂度，而这可通过核技巧来实现，这就是核方法的中心思想。

带核函数的 SMO 算法

Algorithm 3: 带核函数的 SMO 算法

$\lambda = 0, b = 0$

foreach *i, j:* **do**

  $K_{i,j} = K_\phi(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$

for $r = 1, 2, \cdots, N$: **do**

  for $i = 1, 2, \cdots, m$: **do**

    for $j = 1, 2, \cdots, m$: **do**

      $\delta_j = \max\{L_{i,j}, \min\{\lambda_j + \frac{E_j - E_i}{2K_{i,j} - K_{i,i} - K_{j,j}}, H_{i,j}\}\} - \lambda_j$

      $\lambda_j \leftarrow \lambda_j + \delta_j$

      $\lambda_i \leftarrow \lambda_i - y^{(i)} y^{(j)} \delta_j$

      if $\lambda_i > 0$: **then**

        $b = y^{(i)} - \sum_{t=0}^{m} \lambda_t y^{(t)} K_{t,i}$

      else if $\lambda_j > 0$: **then**

        $b = y^{(j)} - \sum_{t=1}^{m} \lambda_t y^{(t)} K_{t,j}$

return $h(\boldsymbol{x}) = Sign(\sum_{t=1}^{m} \lambda_t y^{(t)} K_\phi(\boldsymbol{x}^{(t)}, \boldsymbol{x}) + b)$

核函数的选择对支持向量机算法的实现和效果至关重要，如果核函数选择不适，那么 ϕ 将不能将特征从低维空间投影到线性可分的高维空间。常用的核函数有：

表 1: 常用核函数

名称	表达式	参数
线性核	$K(\boldsymbol{x},\boldsymbol{z})=\boldsymbol{x}^T\boldsymbol{z}$	
多项式核	$K(\boldsymbol{x},\boldsymbol{z})=(\boldsymbol{x}^T\boldsymbol{z}+1)^d$	$d\geq 1$ 为多项式的次数
高斯核	$K(\boldsymbol{x},\boldsymbol{z})=e^{\frac{-\lvert\lvert x-z\rvert\rvert^2}{2\sigma^2}}$	$\sigma>0$ 为高斯核带宽
拉普拉斯核	$K(\boldsymbol{x},\boldsymbol{z})=e^{\frac{-\lvert\lvert x-z\rvert\rvert}{\sigma}}$	$\sigma>0$
Sigmoid 核	$K(\boldsymbol{x},\boldsymbol{z})=\tanh\ (\beta\boldsymbol{x}^T\boldsymbol{z}+\theta)$	$\beta>0,\theta<0$

案例实战 6.5：利用花萼宽和花瓣宽对是否变色鸢尾花分类。
kernel_svm.py,iris_kernel_svm.py

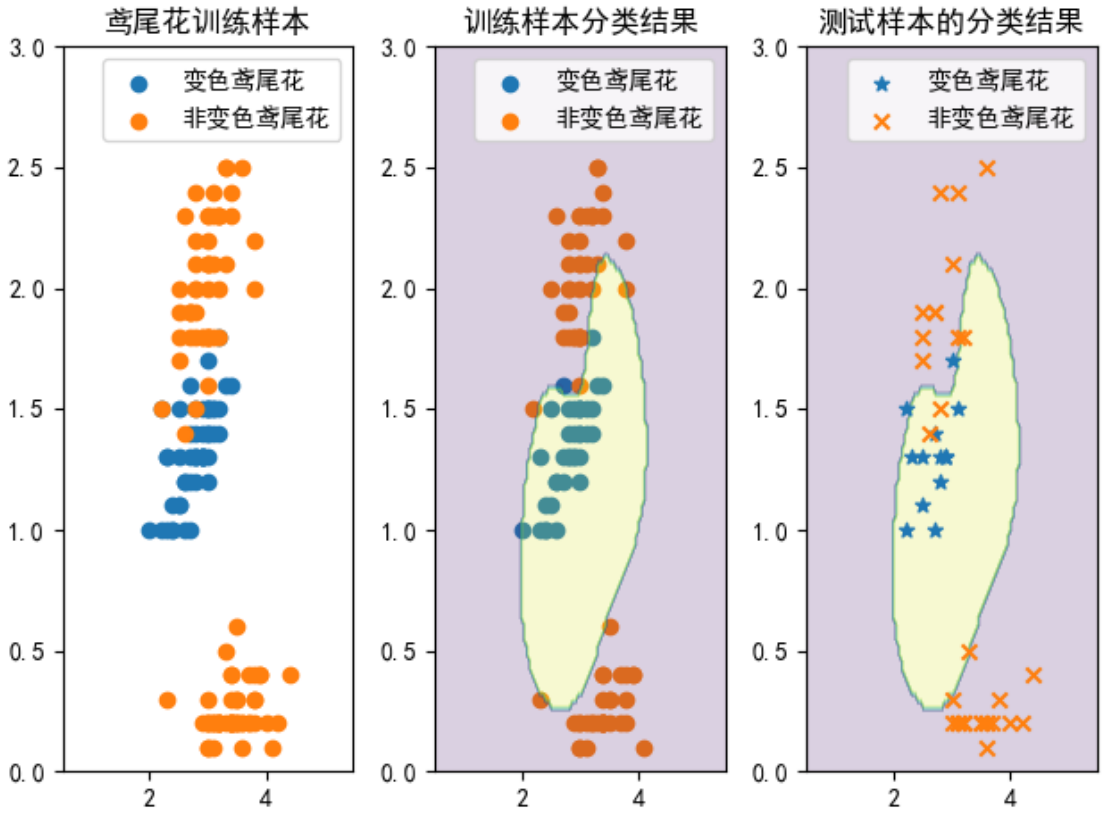


图 4: 高斯核函数支持向量机算法对鸢尾花进行分类

目录

硬间隔支持向量机是一个严格的线性模型，即用一个超平面将数据分隔为两部分。但数据一般情况下很少是严格线性可分的，难以找到一个超平面可以完美地隔开这些数据，如果坚持将所有的数据都做到正确分类，即使在训练模型的时候隔开了，但模型把一些噪声当成了支持向量，那也会得到错误的结果。而采用核方法有时候会导致模型过于复杂，造成过拟合现象。因此，缓解这个问题的另一个方法是允许模型在一些样本上出错，由此，引出软间隔支持向量机的概念。

软间隔是相对于硬间隔定义的。硬间隔要求所有样本必须划分正确的约束条件，即所有样本必须严格满足：$y^{(i)}(\boldsymbol{x}^{(i)T}\boldsymbol{w}+b)\geq 1$。软间隔则是允许某些样本不满足约束条件，因为在样本集中总是存在一些噪声点或者离群点，如果强制要求所有的样本点都满足硬间隔，可能会导致出现过拟合的问题，甚至会使决策边界发生变化，为了避免这个问题的发生，所以在训练过程的模型中，允许部分样本（离群点或者噪声点）不必满足该约束。当然在最大化间隔的同时，不满足约束的样本应尽可能少。放宽约束条件，并将其以惩罚项的形式反映在目标函数中。

软间隔目标函数

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{m}\xi_i \tag{13}$$

$$\text{约束：} y^{(i)}(\boldsymbol{x}^{(i)T}\boldsymbol{w} + b) \geq 1 - \xi_i,\ \xi_i \geq 0, i = 1, 2, \cdots, m \tag{14}$$

约束条件从硬间隔的 ≥ 1 放宽至 $\geq 1 - \xi_i$。同时目标函数比原来多了一个惩罚项 $C\sum_{i=1}^{m}\xi_i$ 。ξ_i 是分离的误差量，ξ_i 越大，惩罚值就越大，$C > 0$ 是算法参数，用以控制惩罚的强度，当 $C \to +\infty$，为了保证取得优化值，必须 $\xi_i = 0$，这时软间隔就变成了硬间隔支持向量机算法。如果训练数据存在分离超平面，则软间隔跟硬间隔将得到相同的分离边界；如果训练数据不存在分离超平面，则硬间隔支持向量机算法无法求解，而软间隔支持向量机计算出一个近似地分离训练数据中正负采样的线性边界。这与核方法得到的非线性边界不同。

案例实战 6.6 弗吉尼亚鸢尾花识别。iris_soft_svm.py
soft_svm_smo.py 实现软间隔支持向量机算法的类

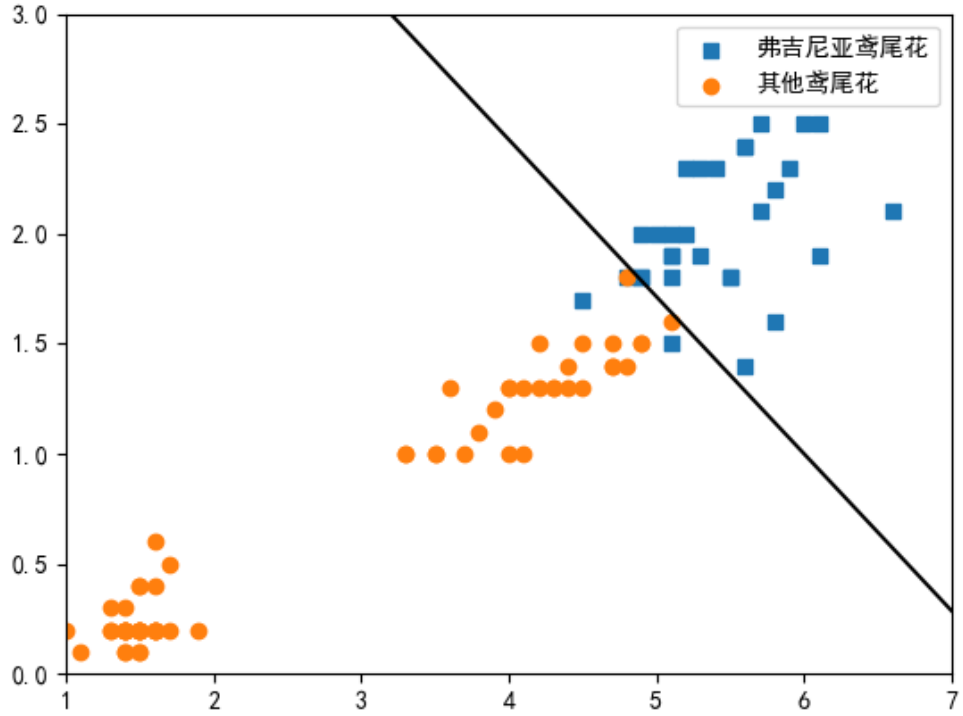


图 5: 软间隔支持向量机算法识别弗吉尼亚鸢尾花

Hinge 损失与软间隔支持向量机

定义

Hinge 损失映射 $\ell : \{-1, 1\} \times \mathbb{R} \to \mathbb{R}$, $\ell(y, z) = \max\{0, 1 - yz\}$ 称为 *Hinge* 损失。其中第一个参数 y 表示正确的类别，标签取值为 $\{-1, +1\}$，第二个参数 z 表示预测输出，是一个实数。

软间隔优化问题（13）可转化为无约束优化问题

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{m} \max\{0, 1 - y^{(i)}(\boldsymbol{x}^{(i)T}\boldsymbol{w} + b)\} \tag{15}$$

该式与 L_2 正则化的目标函数非常相似，将其除以 mC 并不会改变其最优解，得到与其等价的优化问题：

$$\min_{\boldsymbol{w},b} \frac{1}{2mC}||\boldsymbol{w}||^2 + \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y^{(i)}(\boldsymbol{x}^{(i)T}\boldsymbol{w} + b)\} \tag{16}$$

这是一个模型假设为线性模型 $h_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{w} + b$ 且损失函数为 Hinge 损失的经验损失最小化算法的 L_2 正则化，其正则化系数为 $\frac{1}{2mC}$。

因为 Hinge 损失并不是处处可微，因此要使用次梯度下降算法来求解目标函数（16）。对任意一条训练数据 $(\boldsymbol{x}, y)$，$\boldsymbol{x} \in \mathbb{R}^n, y \in \mathbb{R}$，损失函数 $\ell(y, h(\boldsymbol{x})) = \max\{0, 1 - y(\boldsymbol{x}^T\boldsymbol{w} + b)\}$，则 $v_w = -I\{y(\boldsymbol{x}^T\boldsymbol{w} + b) < 1\}y\boldsymbol{x}$ 和 $v_b = -I\{y(\boldsymbol{x}^T\boldsymbol{w} + b) < 1\}y$ 分别是损失函数 ℓ 关于 $\boldsymbol{w}$ 和 b 的次梯度。

软间隔支持向量机的次梯度下降算法 soft_svm_hinge.py，合页损失对弗吉尼亚鸢尾花识别 iris_soft_svm_hinge.py

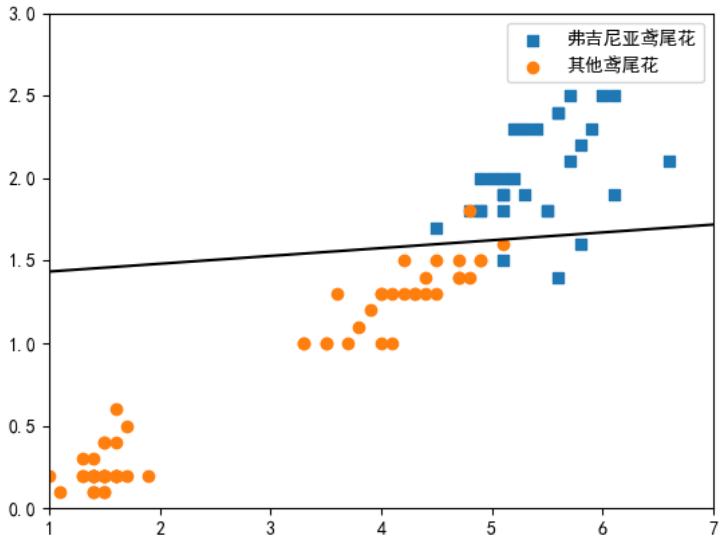


图 6: 合页弗吉尼亚鸢尾花识别

目录

Sklearn 的 SVM 简介

Sklearn 中 SVM 的算法库分为两类，一类是分类的算法库，包括 SVC，NuSVC，和 LinearSVC 3 个类。另一类是回归算法库，包括 SVR，NuSVR，和 LinearSVR 3 个类。相关的类都包含在 sklearn.svm 模块之中。本节只介绍其分类算法。对于 SVC，NuSVC，和 LinearSVC 3 个分类的类，SVC 和 NuSVC 差不多，区别仅仅在于对损失的度量方式不同，而 LinearSVC 从名字就可以看出，它是线性分类，也就是不支持各种低维到高维的核函数，仅仅支持线性核函数，对线性不可分的数据不能使用。使用这些类的时候，如果有经验知道数据是线性可以拟合的，那么使用 LinearSVC 去分类或者 LinearSVR 去回归，它们不需要通过调参去选择各种核函数以及对应参数，速度较快。如果对数据分布没有什么经验，一般使用 SVC 去分类或者 SVR 去回归，就需要选择核函数以及对核函数调参了。如果对训练集训练的错误率或者说支持向量的百分比有要求的时候，可以选择 NuSVC 分类和 NuSVR ，它们有一个参数来控制这个百分比。

LinearSVC 其函数原型如下：class sklearn.svm.LinearSVC(self, penalty='l2', loss='squared_hinge', dual=True, tol=1e-4, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000)

SVM 算法的调参要点

1. 一般推荐在做训练之前对数据进行归一化，当然测试集中的数据也需要归一化。
2. 在特征数量非常多的情况下，或者样本数远小于特征数量的时候，使用线性核，效果已经很好，并且只需要选择惩罚系数 C 即可。
3. 在选择核函数时，如果线性拟合不好，一般推荐使用默认的高斯核'rbf'。这时主要对惩罚系数 C 和核函数参数 γ 进行调参，通过多轮的交叉验证选择合适的惩罚系数 C 和核函数参数 γ。
4. 理论上高斯核不会比线性核差，但要花费更多的时间在调参上，所以实际上能用线性核解决问题尽量使用线性核。

Sklearn 的 SVM 案例

在 SVM 中，其中最重要的就是核函数的选取和参数选择了，这个需要大量的经验来支撑。

案例 6.7 线性可分支持向量机。linear_kernel_sklearn.py, linear_kernel_sklearn_2.py

案例 6.8 线性不可分支持向量机。本案例使用到 sklearn 中的另一个玩具数据集 make_circles，其作用为在二维空间中创建一个包含较小圆的大圆的样本集，用于可视化聚类和分类算法。linear_unseperable_sklearn.py, gaussian_kernel_sklearn.py

案例 6.9 线性近似可分支持向量机——软间隔问题。soft_svm_sklearn.py