SVM自主实践与SKlearn调用

TA:吴韫琛

E-mail:652024330030@smail.nju.edu.cn

实验内容

- 从零开始构建SVM支持向量机算法，对简单鸢尾花数据集进行分类
- 可视化SVM的分类结果
- 调用Sklearn包中封装好的SVM方法直接处理数据

实验目标

- 从代码层面理解SVM的原理算法与实现
- 掌握直接调用现有的SVM方法解决问题

实验代码

导入相关库

```
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

数据集预处理

加载sklearn iris数据集，仅选取前两个特征，并对标签进行转换将为0的改为-1

```
# 加载鸢尾花数据集
iris = datasets.load_iris()
X = iris.data  # 特征
X = X[:, :2]
y = iris.target  # 目标标签

# 选择前两类（类别 0: setosa, 类别 1: versicolor）
selected_classes = [0, 1]
mask = np.isin(y, selected_classes)

X_filtered = X[mask]
y_filtered[y_filtered == 0] = -1

X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered, test_size=0.3,
random_state=1, stratify=y_filtered)
```

可视化SVM分类结果

我们将iris训练集与测试集放在同一张图中呈现，其中测试集用颜色较浅的数据点表示。对于示例实验中的二维数据二分类问题，SVM的分类结果超平面退化成一条直线。如果分类算法实现正确的话，可视化结果将会看到颜色不同的两类点（标签不同）分居直线两侧。

```
# 准备绘制数据点(训练集与验证集)
classified_train_pts = {'+1': [], '-1': []}
for point, label in zip(X_train, y_train):
    if label == 1.0:
        classified_train_pts['+1'].append(point)
    else:
        classified_train_pts['-1'].append(point)

classified_test_pts = {'+1': [], '-1': []}
for point, label in zip(X_test, y_test):
    if label == 1.0:
        classified_test_pts['+1'].append(point)
    else:
        classified_test_pts['-1'].append(point)

def visualize(w, b, title):
    fig = plt.figure()
    ax = fig.add_subplot(111)

    for label, pts in classified_train_pts.items():
        pts = np.array(pts)
        if label == "+1":
            ax.scatter(pts[:, 0], pts[:, 1], label="versicolor_train", c='#F78C60')
        else:
            ax.scatter(pts[:, 0], pts[:, 1], label="setosa_train", c='#8DB1E3')

    for label, pts in classified_test_pts.items():
        pts = np.array(pts)
        if label == "+1":
            ax.scatter(pts[:, 0], pts[:, 1], label="versicolor_test", c='#FEE3C5')
        else:
            ax.scatter(pts[:, 0], pts[:, 1], label="setosa_test", c='#C3D4DE')

    x1, _ = max(X_train, key=lambda x: x[0])
    x2, _ = min(X_train, key=lambda x: x[0])
    a1, a2 = w
    y1, y2 = (-b - a1*x1)/a2, (-b - a1*x2)/a2
    ax.plot([x1, x2], [y1, y2])

    plt.legend()
    plt.title(title)
    plt.show()
```

基于随机梯度下降的SVM算法

1. SVM算法基本原理

我们回顾一下SVM的基本原理，SVM旨在寻找一个能分隔两类数据的超平面，同时最大化两类数据到超平面的间隔。对于本示例代码中处理的二分类问题，假设类别标签 y 为 -1 和 +1，超平面的表达式为：

$$f(x) = W \cdot x + b \tag{1}$$

2. 损失函数与目标函数

SVM的目标函数由两部分组成：

- **正则化项**：控制模型复杂度，形式为 $\frac{1}{2}\|W\|^2$。
- **Hinge Loss 损失项**：衡量分类错误的惩罚，定义为：$L(y, f(x)) = \max(0, 1 - y(W \cdot x + b))$
- 综合目标函数为：

$$\min_{W,b} \quad \frac{1}{2}\|W\|^2 + C\sum_i \max(0, 1 - y_i(W \cdot x_i + b)) \tag{2}$$

  其中 (C) 是平衡正则化与损失的超参数。

3. 梯度下降（SGD）实现

通过随机梯度下降更新参数：

- **当 $y_i(W \cdot x_i + b) \geq 1$ 时**：只考虑正则化项更新权重：

$$W \leftarrow W - \eta \cdot 2\lambda W \tag{3}$$

  偏置 (b) 保持不变。

- **当 $y_i(W \cdot x_i + b) < 1$ 时**：同时更新正则化和损失项：

$$W \leftarrow W - \eta(2\lambda W - y_i \cdot x_i) \tag{4}$$

  同时更新偏置：

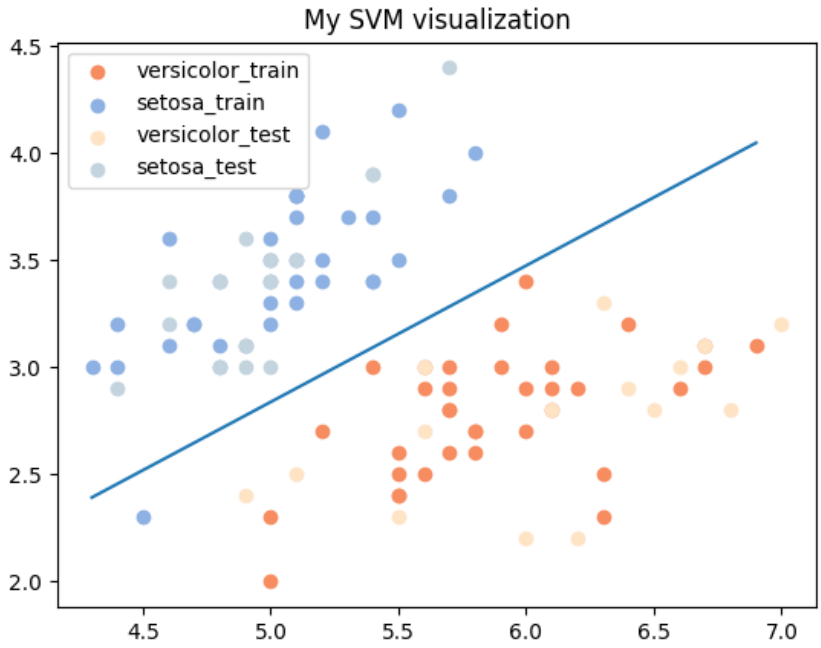
$$b \leftarrow b + \eta \cdot y_i \tag{5}$$

```
# 使用随机梯度下降实现SVM
class LinearSVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, epochs=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.epochs = epochs
        self.W = None
        self.b = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.W = np.zeros(n_features)  # 初始化权重
        self.b = 0  # 初始化偏置项

        for epoch in range(self.epochs):
            for i in range(n_samples):
```

```
                if y[i] * (np.dot(X[i], self.W) + self.b) < 1:
                    self.W -= self.lr * (2 * self.lambda_param * self.W - np.dot(X[i],
y[i]))
                    self.b += self.lr * y[i]
                else:
                    self.W -= self.lr * 2 * self.lambda_param * self.W

    def predict(self, X):
        return np.sign(np.dot(X, self.W) + self.b)
```

```
'''
拟合训练数据
'''
svm = LinearSVM()
svm.fit(X_train, y_train)
# 预测
y_pred = svm.predict(X_test)

# 计算准确率
accuracy = np.mean(y_pred == y_test)
print("SVM 分类器的准确率:", accuracy)

# 绘制训练集数据点
visualize(svm.W, svm.b, "My SVM visualization")
```

4. 实验结果



基于 SMO 算法实现 SVM

我们利用 SMO（Sequential Minimal Optimization）求解 SVM 的对偶问题，方法原理如下所示。

1. SVM 目标与对偶问题

给定样本 $\{(x_i, y_i)\}_{i=1}^m$（其中 $y_i \in \{-1, +1\}$），SVM 寻找一个超平面

$$f(x) = W \cdot x + b \tag{6}$$

以最大化两类间隔，并允许部分误分类（软间隔）。

- 原始问题（软间隔 SVM）

$$\min_{W,b,\xi} \quad \frac{1}{2}\|W\|^2 + C\sum_{i=1}^m \xi_i \tag{7}$$

$$\text{s.t.} \quad y_i(W \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \tag{8}$$

- 对偶问题

$$\max_{\alpha} \quad \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m \alpha_i\alpha_j\, y_i y_j \,\langle x_i, x_j\rangle \tag{9}$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^m \alpha_i y_i = 0 \tag{10}$$

- 求得最优 α_i 后，可计算权重向量：

$$W = \sum_{i=1}^m \alpha_i\, y_i\, x_i \tag{11}$$

2. SMO 算法基本思想

SMO 算法通过反复选取两个 α 同时更新来求解对偶问题，核心步骤包括：

- **分解问题**：每次仅优化一对 α，其他保持不变，从而满足约束 $\sum \alpha_i y_i = 0$。
- **迭代更新**：不断选择不同的 α 对进行更新，直至所有更新满足停止条件。

3. 关键更新公式

- 计算误差与步长
  对于选定的两个样本 i 和 j，计算预测误差：

  $$E_i = f(x_i) - y_i, \quad E_j = f(x_j) - y_j \tag{12}$$

  定义步长：

  $$\eta = K_{ii} + K_{jj} - 2K_{ij} \tag{13}$$

  其中

  $$K_{ii} = \langle x_i, x_i\rangle, \quad K_{jj} = \langle x_j, x_j\rangle, \quad K_{ij} = \langle x_i, x_j\rangle. \tag{14}$$

  当 $\eta \leq 0$ 时，跳过本次更新。
- 更新 α_j 和 α_i

  先更新 α_j：

$$\alpha_j^{new} = \alpha_j + \frac{y_j(E_i - E_j)}{\eta} \tag{15}$$

再通过上下界约束（根据 y_i 与 y_j 的关系）将 α_j^{new} 限制在区间 $[L, H]$ 内：

- 若 $y_i \neq y_j$：

$$L = \max(0,\, \alpha_j - \alpha_i), \quad H = \min(C,\, C + \alpha_j - \alpha_i) \tag{16}$$

- 若 $y_i = y_j$：

$$L = \max(0,\, \alpha_i + \alpha_j - C), \quad H = \min(C,\, \alpha_i + \alpha_j) \tag{17}$$

再利用约束关系更新 α_i：

$$\alpha_i^{new} = \alpha_i + y_i y_j (\alpha_j - \alpha_j^{new}) \tag{18}$$

- 更新阈值 b

计算两个候选阈值：

$$b_i = -E_i - y_i K_{ii}(\alpha_i^{new} - \alpha_i) - y_j K_{ij}(\alpha_j^{new} - \alpha_j) + b \tag{19}$$

$$b_j = -E_j - y_i K_{ij}(\alpha_i^{new} - \alpha_i) - y_j K_{jj}(\alpha_j^{new} - \alpha_j) + b \tag{20}$$

根据 α_i^{new} 和 α_j^{new} 是否在 $(0, C)$ 内选择：

$$b = \begin{cases} b_i, & \text{若 } 0 < \alpha_i^{new} < C \\ b_j, & \text{若 } 0 < \alpha_j^{new} < C \\ \frac{b_i + b_j}{2}, & \text{否则} \end{cases} \tag{21}$$

4. 算法流程概述

使用 SMO 算法求解 SVM 对偶问题的基本流程是通过局部两变量优化满足全局约束，逐步逼近最优解。

- **初始化**：令所有 $\alpha_i = 0$ 和 $b = 0$。
- **迭代更新**：对每个样本计算误差 E_i，随机选取另一个样本 j 计算 E_j，更新 α_j 和 α_i，并更新阈值 b。
- **终止条件**：当连续迭代中无有效更新达到预设次数时停止。
- **构造模型**：最终利用

$$W = \sum_{i=1}^{m} \alpha_i\, y_i\, x_i \tag{22}$$

得到分类器

$$f(x) = W \cdot x + b \tag{23}$$

```
'''
smo算法辅助函数
'''
def clip(alpha, L, H):
    ''' 修建alpha的值到L和H之间.
    '''
    if alpha < L:
```

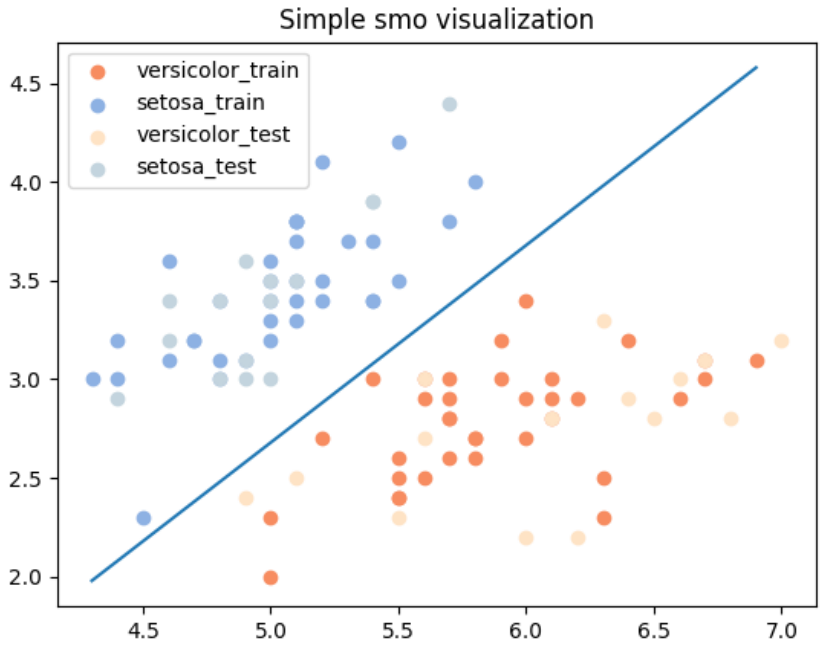
```
        return L
    elif alpha > H:
        return H
    else:
        return alpha
def select_j(i, m):
    ''' 在m中随机选择除了i之外剩余的数
    '''
    l = list(range(m))
    seq = l[: i] + l[i+1:]
    return np.random.choice(seq)

def get_w(alphas, dataset, labels):
    ''' 通过已知数据点和拉格朗日乘子获得分割超平面参数w
    '''
    alphas, dataset, labels = np.array(alphas), np.array(dataset), np.array(labels)
    yx = labels.reshape(1, -1).T*np.array([1, 1])*dataset
    w = np.dot(yx.T, alphas)
    return w.tolist()
```

```
'''
简单版本smo算法实现
'''
def simple_smo(dataset, labels, C, max_iter):
    ''' 简化版SMO算法实现，未使用启发式方法对alpha对进行选择.
    :param dataset: 所有特征数据向量
    :param labels: 所有的数据标签
    :param C: 软间隔常数, 0 <= alpha_i <= C
    :param max_iter: 外层循环最大迭代次数
    '''
    dataset = np.array(dataset)
    m, n = dataset.shape
    labels = np.array(labels)
    # 初始化参数
    alphas = np.zeros(m)
    b = 0
    it = 0
    def f(x):
        "SVM分类器函数 y = w^Tx + b"
        # Kernel function vector.
        x = np.matrix(x).T
        data = np.matrix(dataset)
        ks = data*x
        # Predictive value.
        wx = np.matrix(alphas*labels)*ks
        fx = wx + b
        return fx[0, 0]

    while it < max_iter:
        pair_changed = 0
        for i in range(m):
```

```
            a_i, x_i, y_i = alphas[i], dataset[i], labels[i]
            fx_i = f(x_i)
            E_i = fx_i - y_i
            j = select_j(i, m)
            a_j, x_j, y_j = alphas[j], dataset[j], labels[j]
            fx_j = f(x_j)
            E_j = fx_j - y_j
            K_ii, K_jj, K_ij = np.dot(x_i, x_i), np.dot(x_j, x_j), np.dot(x_i, x_j)
            eta = K_ii + K_jj - 2*K_ij
            if eta <= 0:
                continue
            # 获取更新的alpha对
            a_i_old, a_j_old = a_i, a_j
            a_j_new = a_j_old + y_j*(E_i - E_j)/eta
            # 对alpha进行修剪
            if y_i != y_j:
                L = max(0, a_j_old - a_i_old)
                H = min(C, C + a_j_old - a_i_old)
            else:
                L = max(0, a_i_old + a_j_old - C)
                H = min(C, a_j_old + a_i_old)
            a_j_new = clip(a_j_new, L, H)
            a_i_new = a_i_old + y_i*y_j*(a_j_old - a_j_new)
            if abs(a_j_new - a_j_old) < 0.00001:
                continue
            alphas[i], alphas[j] = a_i_new, a_j_new
            # 更新阈值b
            b_i = -E_i - y_i*K_ii*(a_i_new - a_i_old) - y_j*K_ij*(a_j_new - a_j_old) + b
            b_j = -E_j - y_i*K_ij*(a_i_new - a_i_old) - y_j*K_jj*(a_j_new - a_j_old) + b
            if 0 < a_i_new < C:
                b = b_i
            elif 0 < a_j_new < C:
                b = b_j
            else:
                b = (b_i + b_j)/2
            pair_changed += 1
        if pair_changed == 0:
            it += 1
        else:
            it = 0
    return alphas, b
```

```
# 使用简化版SMO算法优化SVM
alphas, b = simple_smo(X_train, y_train, 0.6, 40)
w = get_w(alphas, X_train, y_train)

visualize(w, b, "Simple smo train set visualization")
```

5. 实验结果



调用Sklearn内部的SVM方法

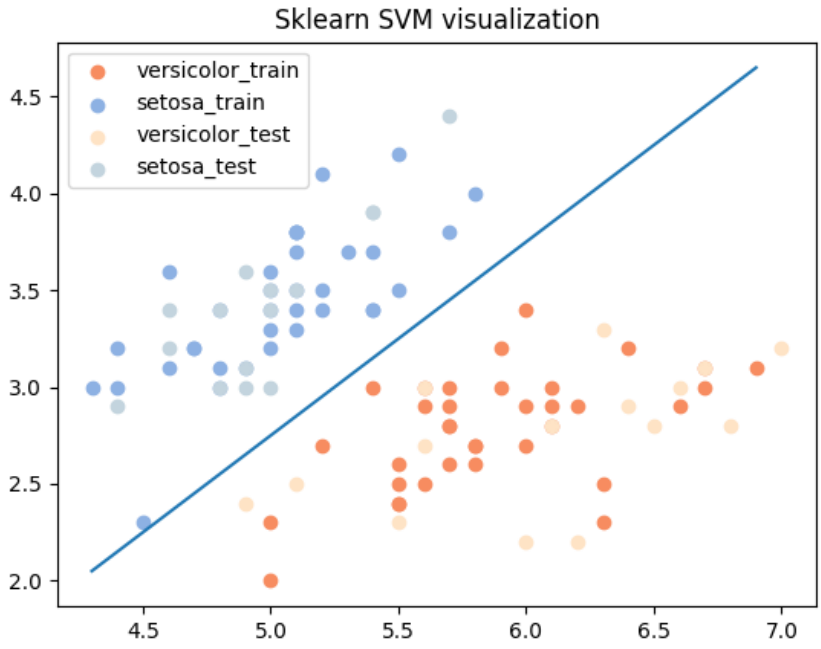
这一部分我们直接使用Sklearn封装的SVM方法，并与上述两种方法进行可视化对比。其底层实现我们留作课后思考，希望同学们去探索。关于Sklearn封装的各种SVM方法以及相应参数的含义我们整理在下表中：

方法	主要参数及含义	适用任务及特点
SVC	- **C**: 正则化参数，控制误分类惩罚 - **kernel**: 核函数类型（如 'linear', 'poly', 'rbf', 'sigmoid'） - **degree**: 多项式核函数的度数（仅当 kernel='poly' 时有效） - **gamma**: 核系数（适用于 'rbf', 'poly', 'sigmoid'） - **coef0**: 核函数中的常数项（用于 'poly' 和 'sigmoid'） - **shrinking**: 是否启用收缩启发式 - **probability**: 是否启用概率估计 - **tol**: 停止准则容忍度 - **max_iter**: 最大迭代次数	适用于分类任务，支持多种核函数，适合处理非线性数据。
NuSVC	- **nu**: 控制误差比例和支持向量比例（上界训练误差，下界支持向量比例） - 其他参数同 SVC（kernel, degree, gamma, coef0, shrinking, probability, tol, max_iter）	类似于 SVC，但使用参数 v 控制训练误差和支持向量的比例，提供另一种正则化方式。
	- **C**: 正则化参数 - **penalty**: 惩罚项类型（'l2' 或 'l1'） - **loss**: 损失函数（'hinge' 或	

LinearSVC	'squared_hinge'） - **dual**: 是否使用对偶问题求解 - **tol**: 停止准则容忍度 - **max_iter**: 最大迭代次数 - **multi_class**: 多分类策略（如 'ovr' 或 'crammer_singer'）	适用于线性可分的分类任务，适合大规模数据集，计算效率高。
SVR	- **C**: 正则化参数 - **kernel**: 核函数类型 - **degree**: 多项式核函数的度数（当 kernel='poly' 时有效） - **gamma**: 核系数 - **coef0**: 核函数中的常数项 - **epsilon**: ε-不敏感损失中的 ε 值 - **tol**: 停止准则容忍度 - **max_iter**: 最大迭代次数	适用于回归任务，支持多种核函数，能够处理非线性回归问题。
NuSVR	- **nu**: 控制误差比例（与 SVR 类似，但使用 ν 参数） - 其他参数同 SVR（C, kernel, degree, gamma, coef0, tol, max_iter）	类似于 SVR，但使用参数 ν 控制训练误差和支持向量的比例，提供另一种正则化方式。
LinearSVR	- **C**: 正则化参数 - **epsilon**: ε-不敏感损失中的 ε 值 - **loss**: 损失函数（'epsilon_insensitive' 或 'squared_epsilon_insensitive'） - **dual**: 是否使用对偶求解 - **tol**: 停止准则容忍度 - **max_iter**: 最大迭代次数	适用于线性回归任务，计算效率高，适合大规模数据集。
OneClassSVM	- **nu**: 上界训练错误比例和下界异常样本比例 - **kernel**: 核函数类型 - **degree**, **gamma**, **coef0**: 与其他 SVM 类似，用于核函数 - **tol**: 停止准则容忍度 - **max_iter**: 最大迭代次数	主要用于异常值检测，适合在无标签数据中识别异常或新颖模式。

```
from sklearn.svm import SVC

svm_sklearn = SVC(kernel='linear')
svm_sklearn.fit(X_train, y_train)

y_pred_sklearn = svm_sklearn.predict(X_test)

# 计算准确率
accuracy = np.mean(y_pred_sklearn == y_test)
print("SVM 分类器的准确率:", accuracy)

# # 绘制训练集和测试集数据点
visualize(svm_sklearn.coef_[0], svm_sklearn.intercept_, "Sklearn SVM train set
visualization")
```

实验结果



课后问题

- sklearn iris是一个三分类数据集，在我们的示例代码中仅仅取了两类作二分类实验，请实现对于三分类iris的SVM算法，并探索可视化方案
- 对于iris这样线性可分的简单数据集几种方法的准确率均为100%，请尝试更换其他复杂的线性不可分数据集，调用SKlearn中的不同kernel进行实验，并进行性能评估（准确率，拟合时间等指标）
- 调研sklearn内部集成的SVM方法的底层实现，思考为什么采取这种方案。（是否最优？或是准确率与效率的平衡？）