SVM 实验报告

王昊然 PB20010382

2023 年 11 月 25 日

1 简介

本次实验旨在探究支持向量机（SVM）二分类器在不同算法下的表现。实验中分别采用了序列最小优化（SMO）算法和梯度下降 (GD) 算法来解决 SVM 的对偶问题，并比较这两种方法在分类准确率和计算时间上的差异。此外，为了提供一个参考标准，我们还利用了 sklearn 库的 SVM 实现进行了同样的分类任务，以便与自行实现的算法进行对比。

实验所用数据集由助教提供的数据生成函数生成，旨在有效地测试不同算法在处理二分类问题时的表现。更多关于实验要求和框架的详细信息，可以参考以下链接：github 实验要求和框架。

2 理论背景

2.1 SVM 的基本理论

2.1.1 思想

支持向量机（SVM）是一种有效的分类方法，特别适用于二分类问题。其基本思想是找到一个能够正确分类两类数据的最优决策平面（或超平面），同时最大化该平面到最近训练样本点（支持向量）的距离。这个距离被称为间隔。最优决策平面的数学表示为：

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

其中，$\mathbf{w}$ 是超平面的法向量，$\mathbf{x}$ 是数据点，b 是偏置项。

2.1.2 原问题的公式表示

在 SVM 中，我们希望找到参数 $\mathbf{w}$ 和 b，以最大化间隔的同时正确分类所有数据点。这可以通过解决以下优化问题实现：

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

$$\text{s.t. } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \forall i$$

这里，$\|\mathbf{w}\|^2$ 代表了间隔的倒数，因此最小化 $\|\mathbf{w}\|^2$ 实际上是在最大化间隔。$y_i \in \{-1, 1\}$ 是第 i 个数据点的类别标签。

2.1.3 一般有约束最优化问题的 Lagrange 函数和对偶问题

在最优化理论中，构造 Lagrange 函数来求解，Lagrange 函数将原问题中的约束条件以乘子的形式结合到目标函数中，最后转化为对偶问题是常见的套路。

假设我们有以下一般形式的优化问题：

$$\min f(\mathbf{x})$$

$$\text{s.t. } g_i(\mathbf{x}) \leq 0, \forall i \in \{1, \ldots, m\}$$

$$h_j(\mathbf{x}) = 0, \forall j \in \{1, \ldots, p\}$$

其中，$f(\mathbf{x})$ 是目标函数，$g_i(\mathbf{x})$ 和 $h_j(\mathbf{x})$ 分别是不等式和等式约束。

对应的 Lagrange 函数 $L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ 定义为：

$$L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \sum_{i=1}^{m} \alpha_i g_i(\mathbf{x}) + \sum_{j=1}^{p} \beta_j h_j(\mathbf{x})$$

其中，$\boldsymbol{\alpha} \succeq 0$ 和 $\boldsymbol{\beta}$ 是 Lagrange 乘子，对应于每个不等式和等式约束。

在得到 Lagrange 函数后，对偶问题的形成涉及到最大化 $L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ 对于乘子的最小值。对偶问题可以表达为：

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\beta}} \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

$$\text{s.t. } \alpha_i \geq 0, \forall i$$

对偶问题的求解通常更简单，因为它减少了原问题的约束数量。

2.1.4 Lagrange 乘子法与对偶问题

对于 SVM 的原问题，采用 Lagrange 乘子法。通过引入一组 Lagrange 乘子 $\alpha_i \geq 0$，原问题转化为 Lagrange 函数 $L(\mathbf{w}, b, \boldsymbol{\alpha})$：

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1]$$

2.1.5 SVM 的对偶问题

对于 SVM 而言，对偶问题是通过最大化 Lagrange 函数 $L(\mathbf{w}, b, \boldsymbol{\alpha})$ 相对于 $\boldsymbol{\alpha}$ 的最小值得到的。对偶问题的优势在于它通常更容易求解，对偶问题的公式表示为：

$$\max_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j < \mathbf{x}_i, \mathbf{x}_j > \right)$$

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

这里，$\boldsymbol{\alpha}$ 是 Lagrange 乘子的向量，它的每个元素对应于训练数据集中的一个样本。

2.2 软间隔 SVM

在实际应用中，数据往往不是完全线性可分的，这时软间隔 SVM 更为适用。(例如在本次实验生成数据中有一些 mislabel 的情况下)。

2.2.1 原问题

软间隔 SVM 的原始优化问题引入了松弛变量（slack variables）$\xi_i \geq 0$，用于处理数据点可能不完全满足间隔要求的情况。原问题可以表示为：

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i$$
$$\text{s.t. } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \forall i$$
$$\xi_i \geq 0, \forall i$$

这里，C 是一个正则化参数，控制着间隔宽度与分类错误之间的权衡。

2.2.2 Lagrange 函数

对于软间隔 SVM 的原问题，其 Lagrange 函数 $L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu})$ 可以定义为：

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{n} \mu_i \xi_i$$

其中，$\boldsymbol{\alpha}$ 和 $\boldsymbol{\mu}$ 是对应于不等式约束的 Lagrange 乘子。

2.2.3 对偶问题

软间隔 SVM 的对偶问题可以通过最大化 Lagrange 函数 $L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu})$ 相对于乘子的最小值得到。对偶问题表达为：

$$\max_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j < \mathbf{x}_i, \mathbf{x}_j > \right)$$

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

$$\mu_i = C - \alpha_i, \forall i$$

在这个对偶问题中，我们通过调整 Lagrange 乘子 $\boldsymbol{\alpha}$ 来最大化目标函数，同时满足相应的约束条件。这种形式的对偶问题使得算法能够更灵活地处理非线性可分数据，同时考虑到噪声和异常值的存在。

3 实验方法

3.1 SMO 算法

3.1.1 数学原理

SM 算法是解决 SVM 训练问题的一种有效方法，特别是在处理大规模数据集时。SMO 算法的核心在于将大型的二次规划（QP）问题分解为一系列小型的二次规划问题。这些小型的二次规划问题针对 SVM 的一对 Lagrange 乘子进行优化，每次迭代只优化两个乘子，使得问题规模大大减小。

SVM 的对偶问题：

$$\max_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j < \mathbf{x}_i, \mathbf{x}_j > \right)$$

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

$$0 \leq \alpha_i \leq C, \forall i$$

SMO 算法通过迭代地选择一对 Lagrange 乘子 α_i 和 α_j 并优化它们，同时保持其他所有乘子固定。选择乘子时，SMO 尝试挑选出违反 KKT 条件最严重的乘子，从而加速收敛过程。

在每次迭代中，算法首先计算乘子的解析解。解的计算涉及到对应样本的核函数值，以及当前模型的误差。更新的公式为：

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

其中，$\eta = K_{ii} + K_{jj} - 2K_{ij}$，$E_i$ 和 E_j 是样本 i 和 j 的预测误差。

由于解必须满足一定的约束条件，因此在求得解析解后，SMO 算法通过裁剪步骤将乘子的值调整到合法范围内。

3.1.2 终止条件

SMO 算法的终止条件是基于 KKT 条件的满足程度。KKT 条件是求解约束优化问题的必要条件，具体到 SVM 的场景，这些条件可以表达为：

$$0 \leq \alpha_i \leq C$$

$$y_i f(x_i) - 1 \geq 0, \text{ if } \alpha_i = 0$$

$$y_i f(x_i) - 1 = 0, \text{ if } 0 < \alpha_i < C$$

$$y_i f(x_i) - 1 \leq 0, \text{ if } \alpha_i = C$$

其中，$f(x_i)$ 是对样本 x_i 的预测值，α_i 是 Lagrange 乘子。互补松弛条件，即 α_i 与 $y_i f(x_i) - 1$ 乘积为零，是判断 KKT 条件满足程度的关键部分。

3.1.3 迭代算法 (具体请见统计机器学习一书)

SMO 算法的伪代码如下：

1. 初始化：设定 $C, tol = 1, 1e-5$。
2. 初始化 $\boldsymbol{\alpha}, coef, intercept$。
3. 检查是否所有样本满足 KKT 条件：
   - 若满足，则算法停止。
   - 否则，进入下一步。
4. 选择违反 KKT 条件最严重的样本 i, 注意：如果有多个 “最”, 随机选取其中之一即可。并选择 j 使得 $|E_i - E_j|$ 最大, 其中 $E_k = \text{func}(coef, intercept, X[k,:]) - y[k]$。注意：如果有多个 “最”，随机选取其中之一即可。func 函数为决策平面函数
5. 更新 α_i 和 α_j：

$$\alpha_j^{new} = \text{clip}\left(\alpha_j + \frac{y_j(E_i - E_j)}{K_{ii} + K_{jj} - 2K_{ij}}, \text{low}, \text{high}\right)$$

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

   注意到 $y_i \in \{-1, 1\}$, 于是乘 y 和除以 y 没区别

   其中，K_{ij} 表示核函数计算的结果，low 和 high 的计算依赖于 y_i 和 y_j 是否相同。clip 为剪裁函数。对于 low 和 high

   当样本标签 y_i 和 y_j 相同时（即 $y_i = y_j$）：

$$\text{low} = \max(0, \alpha_i + \alpha_j - C)$$

$$\text{high} = \min(C, \alpha_i + \alpha_j)$$

   当样本标签 y_i 和 y_j 不同时（即 $y_i \neq y_j$）：

$$\text{low} = \max(0, \alpha_j - \alpha_i)$$

$$\text{high} = \min(C, C + \alpha_j - \alpha_i)$$

6. 更新 $intercept$：

$$b'_i = b_{\text{old}} - E_i - y_i(\alpha'_i - \alpha_i)K(x_i, x_i) - y_j(\alpha'_j - \alpha_j)K(x_i, x_j)$$
$$b'_j = b_{\text{old}} - E_j - y_i(\alpha'_i - \alpha_i)K(x_i, x_j) - y_j(\alpha'_j - \alpha_j)K(x_j, x_j)$$

根据 α_i 和 α_j 的值，选择 b：

- 如果 $0 < \alpha_i < C$，则使用 b'_i。
- 如果 $0 < \alpha_j < C$，则使用 b'_j。
- 如果 α_i 和 α_j 都在边界值（即 0 或 C），则取 b'_i 和 b'_j 的平均值。

7. 更新 $coef$ 和 $intercept$ 为新值。

8. 重复上述步骤直至满足停止条件。

3.2 梯度下降法

SVM 对偶问题的目标函数可以表示为：

$$g(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

。

对 $g(\boldsymbol{\alpha})$ 关于 α_k 的梯度可以计算如下：

$$\frac{\partial g}{\partial \alpha_k} = 1 - \sum_{i=1}^{n} y_k y_i \alpha_i \langle \mathbf{x}_k, \mathbf{x}_i \rangle$$

。

3.2.1 更新过程

为了最大化 $g(\boldsymbol{\alpha})$，更新过程可以描述如下：

$$\alpha_k^{new} = \alpha_k + lr \times \frac{\partial g}{\partial \alpha_k}$$

其中，lr 是学习率，它决定了在梯度方向上的更新步长。注意，写代码时可能会用到 lr_decay

4 实验结果

下面给出实验结果和部分预测值与原始标签的对比

```
model1:SMO algorithm
train set accuracy:
0.9205
infact mislabeled num:636,train set num:8000
test set accuracy:
0.9205
infact mislabeled num:159,test set num:2000
```

图 1: SMO 正确率

1-10 2000 rows × 2 columns pd.DataFrame

	y_pred	y
0	-1.0	-1.0
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	-1.0	-1.0
5	-1.0	-1.0
6	1.0	1.0

图 2: 部分预测结果和原标签的对比

4.1 SMO 算法的结果

数据集为 20 features; 10000 samples

4.2 GD 算法的结果

数据集为 20 features; 10000 samples

```
model2:Gradient Descent algorithm
train set accuracy:
0.94825
infact mislabeled num:414,train set num:8000
test set accuracy:
0.9385
infact mislabeled num:123,test set num:2000
```

图 3: GD 正确率

1-10 2000 rows × 2 columns pd.DataFrame

	y_pred	y
0	-1.0	-1.0
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	-1.0	-1.0
5	-1.0	-1.0
6	1.0	1.0

图 4: 部分预测结果和原标签的对比

4.3 与 sklearn 的比较

数据集为 20 features; 10000 samples

```
clf.coef_=[[ 0.02274071  0.09404098 -0.1064336   0.00202636  0.18287085  0.1882266
  -0.17106342 -0.1420816  -0.2126855  -0.03034664 -0.22178982 -0.01407329
   0.06690528 -0.10493372  0.08157887  0.04348746 -0.22365544  0.01925659
  -0.11411043 -0.03441737]]
clf.intercept_=[-9.99186188e-05]
testset accuracy:0.951
```

图 5: sklearn 正确率

1-10 1318 rows × 20 columns pd.DataFrame CSV

	0	1	2	3	4	5	6	7	8	
0	-2.223281	-2.007581	1.865614	4.100516	1.982997	1.190086	-6.706623	3.775638	1.218213	11.2
1	-4.008782	8.240056	-5.623054	19.548781	-13.319517	-17.606886	-16.507213	-8.905556	-11.191154	19.5
2	0.722519	10.097873	-15.569416	-6.124421	-1.393518	-7.285375	5.311638	0.040008	3.212659	-7.2
3	-2.222195	9.083397	-1.586068	6.953060	-1.142183	-1.897672	12.591335	-7.519774	-2.830529	-12.9
4	0.663927	5.406022	-13.188968	8.454264	1.310922	3.490837	4.040681	5.124341	11.203618	8.6
5	-12.178461	0.576921	6.083554	-3.516919	3.686657	1.046323	-3.442891	-0.608175	-1.950209	-19.7
6	-4.535031	-8.209054	9.402189	-14.112935	8.841478	4.632172	5.659732	-6.881593	4.753405	17.5

图 6: sklearn 求出的 supporting vectors

4.4 用时对比

对比的结果如下：

算法	正确率	用时
SMO	92.2%	45s
GD	93.8%	0.43s
sklearn	95.1%	22s

5 结论

本次实验深入探讨了支持向量机（SVM）在不同优化算法下的性能表现。我们分别实现了序列最小优化（SMO）算法和梯度下降（GD）算法，并与 sklearn 库的 SVM 实现进行了对比。实验结果表明，算法 SMO 在准确率方面表现最优，但在计算时间上较长。相比之下，算法 GD 虽然在准确率上方差很大，好的步长选取比较困难，但在运算速度上显著快于其他算法。此外，对于库 sklearn, 他的算法正确率均值高，方差小，时间控制也非常好。

此外，我们还观察到在实现 SVM 时，在处理不完全线性可分的数据时，软间隔 SVM 的应用对于提高模型的泛化能力非常有效。同时对于 svm 算法而言，有一个技巧：选取最违背的 index 时，是从“最违背的集合”里随机选取的。

最后，sklearn 库真厉害！