Machine Learning Lab2 SVM

袁雨 PB20151804

一、实验内容与提示

1. 任务：

你需要去完成类 `SVM1` 和 `SVM2` ，并且使用不同的算法去寻找支持向量机的解。 更具体地说，因为解决支持向量机的关键在于解决书本上的二次规划问题（6.6），你只需要使用两种不同的方法去解决（6.6）。剩下的部分，比如预测，内容可以相同。

在完成了类方法的部分之后，你需要测试你代码的效率。比较应当包含以下内容：

1. 正确率，
2. 计算（训练）的时间消耗。

如果可能的话，你可以使用 `sklearn` 与你的代码比较。如果比不过它，也是没事的。

2. 提示：

1. 我们不推荐你使用已有的库函数去**直接**解决二次规划问题，这是会被扣除一部分分数的。当然，如果你无法使用两种方法去解决，你也可以使用库函数。
2. 我们推荐你使用合适的维度去训练、测试，这会使你的结果更加可靠。同时，不同的维度和样本数也会使你的报告内容更丰富。但是不要让他过于冗杂。
3. 因为我们的数据是基于线性核生成的，你不需要尝试其他的核函数。但是你可以使用软间隔或者正则化等方法来提升你模型的能力。切记，这不是本实验的核心内容。
4. 记得添加你的**错标率**，它会由函数 `generate_data` 生成。

二、实验要求

- 禁止使用 `sklearn` 或者其他的机器学习库，你只被允许使用 `numpy` , `pandas` , `matplotlib` , 和 [Standard Library](), 你需要从头开始编写这个项目。
- 你可以和其他同学讨论，但是你不可以剽窃代码，我们会用自动系统来确定你的程序的相似性，一旦被发现，你们两个都会得到这个项目的零分。

三、实验设备和环境

1. 实验设备

设备：HUAWEI MateBook X Pro

处理器：Intel(R) Core(TM) i5-10210U CPU @1.60GHz 2.11 GHz

2. 实验环境

pycharm, jupyter, python 3.10

四、实验原理

1. SVM的基本型

$$\begin{array}{ll}\min_{\boldsymbol{w},b} & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{s.t.} & y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b\right)\geqslant 1, \quad i=1,2,\ldots,m.\end{array}$$

通过求解上式来得到大间隔划分超平面所对应的模型：$f(\boldsymbol{x})=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b$，其中 $\boldsymbol{w}$ 和 b 是模型参数。

2. SVM的对偶问题

对SVM问题的基本型使用拉格朗日乘子法可得到其“对偶问题”，对每条约束添加拉格朗日乘子 $\alpha_i\geqslant 0$，得到该问题的拉格朗日函数：

$$L(\boldsymbol{w},b,\boldsymbol{\alpha})=\frac{1}{2}\|\boldsymbol{w}\|^2+\sum_{i=1}^{m}\alpha_i\left(1-y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b\right)\right)$$

其中 $\boldsymbol{\alpha}=(\alpha_1;\alpha_2;\ldots;\alpha_m)$。令 $L(\boldsymbol{w},b,\boldsymbol{\alpha})$ 对 $\boldsymbol{w}$ 和 b 的偏导为零可得：

$$\boldsymbol{w}=\sum_{i=1}^{m}\alpha_i y_i\boldsymbol{x}_i,$$
$$0=\sum_{i=1}^{m}\alpha_i y_i$$

代入$L(\boldsymbol{w},b,\boldsymbol{\alpha})$得:

$$\arg\max_{\alpha}\sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j x_i^T x_j^T$$
$$s.t.\,\alpha_i\geq 0,\sum_{i=1}^{N}\alpha_i y_i=0$$

这是一个二次规划问题，可使用通用的二次规划算法来求解，然而该问题的规模正比于训练样本数，这会在实际任务中造成很大的开销。为了避开这个障碍， 人们通过利用问题本身的特性，提出了很多高效算法，SMO(Sequential Minimal Optimization）是其中一个著名的代表。

3. SMO算法解二次规划问题

SMO的基本思路是先固定α_i之外的所有参数，然后求α_i上的极值。由于存在约束$\sum_{i=1}^{N}\alpha_i y_i=0$，若固定$\alpha_i$之外的其他变量，则$\alpha_i$可由其他变量导出。于是SMO每次选择两个变量$\alpha_i$和$\alpha_j$，并固定其他参数，这样，在参数初始化后，SMO不断执行如下两个步骤直至收敛：

- 选取一对需更新的变量α_i和α_j；
- 固定α_i和α_j以外的参数，求解对偶问题目标函数获得更新后的α_i和α_j。

  注意到只需选取的α_i和α_j中有一个不满足KKT条件（如下），目标函数就会在迭代后减小。

$$
\begin{cases}\alpha_i \geqslant 0 \\ y_i f(\boldsymbol{x}_i)-1 \geqslant 0 \\ \alpha_i\left(y_i f(\boldsymbol{x}_i)-1\right)=0\end{cases}
$$

下面进行具体求解。

记$K_{i,j}=x_i^T x_j^T$，常数项C表示与α_1,α_2无关的项。

$$
W(\alpha_1,\alpha_2)=\alpha_1+\alpha_2-\frac{1}{2}K_{1,1}y_1^2\alpha_1^2-\frac{1}{2}K_{2,2}y_2^2\alpha_2^2-K_{1,2}y_1y_2\alpha_1\alpha_2-y_1\alpha_1\sum_{i=3}^{N}\alpha_iy_iK_{i,1}-y_2\alpha_2\sum_{i=3}^{N}\alpha_iy_iK_{i,2}+C
$$

根据约束条件 $\sum_{i=1}^{N}\alpha_iy_i=0$ 可以得到 α_1 与 α_2 的关系：$\alpha_1y_1+\alpha_2y_2=-\sum_{i=3}^{N}\alpha_iy_i=\zeta$ 两边同时乘上 y_1，由于 $y_iy_i=1$ 得到：$\alpha_1=\zeta y_1-\alpha_2y_1y_2$

令 $v_1=\sum_{i=3}^{N}\alpha_iy_iK_{i,1},v_2=\sum_{i=3}^{N}\alpha_iy_iK_{i,2}$，将$\alpha_1$ 的表达式代入得到：

$$
W(\alpha_2)=-\frac{1}{2}K_{1,1}(\zeta-\alpha_2y_2)^2-\frac{1}{2}K_{2,2}\alpha_2^2-y_2(\zeta-\alpha_2y_2)\alpha_2K_{1,2}-v_1(\zeta-\alpha_2y_2)-v_2y_2\alpha_2+\alpha_1+\alpha_2+C
$$

我们需要对这个一元函数求极值，令W 对 α_2 的一阶导数为 0 得：

$$
\frac{\partial W(\alpha_2)}{\partial\alpha_2}=-(K_{1,1}+K_{2,2}-2K_{1,2})\alpha_2+K_{1,1}\zeta y_2-K_{1,2}\zeta y_2+v_1y_2-v_2y_2-y_1y_2+y_2^2=0
$$

记 E_i 为 SVM预测值与真实值的误差，即 $E_i=g(x_i)-y_i=\sum_{i=1}^{N}y_j\alpha_jK(x_i,x_j)+b^{\text{new}}-y_i$，记 $\eta=K_{1,1}+K_{2,2}-2K_{1,2}$，

得最终的一阶导数表达式：

$$
\frac{\partial W(\alpha_2)}{\partial\alpha_2}=-\eta\alpha_2^{\text{new}}+\eta\alpha_2^{\text{old}}+y_2(E_1-E_2)=0
$$

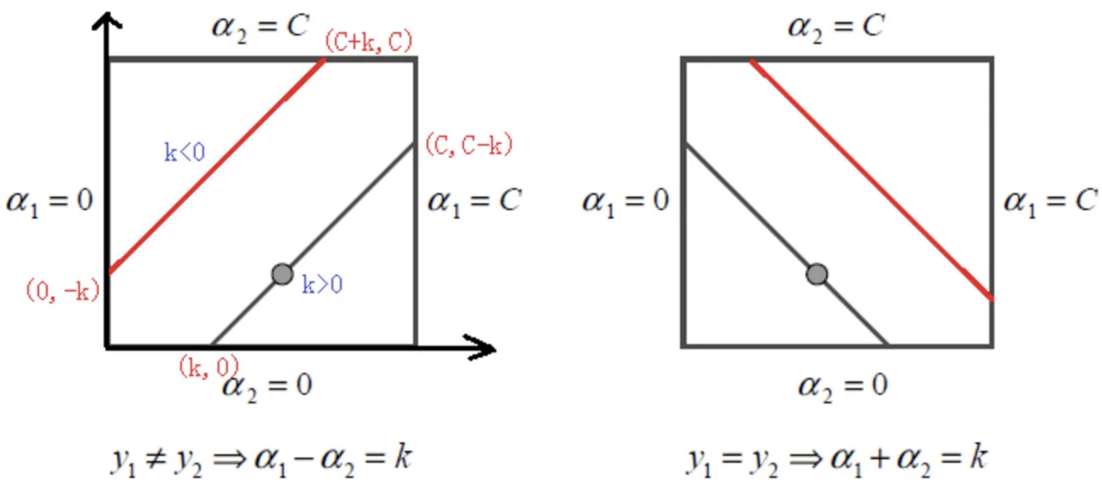
解得：

$$
\alpha_2^{\text{new}}=\alpha_2^{\text{old}}+\frac{y_2(E_1-E_2)}{\eta}
$$

通过对一元函数求极值的方式得到的最优α_i和α_j是未考虑约束条件下的最优解，下面对原始解进行修剪，更正上部分得到的α_2^{new} 为 $\alpha_2^{\text{new, unclipped}}$，即$\alpha_2^{\text{new, unclipped}}=\alpha_2^{\text{old}}+\frac{y_2(E_1-E_2)}{\eta}$。

但在SVM中 α_i 是有约束的，即$\alpha_1y_1+\alpha_2y_2=-\sum_{i=3}^{N}\alpha_iy_i=\zeta$，$0\leq\alpha_i\leq C$。

在二维平面中我们可以看到这是个限制在方形区域中的直线 (见下图)。



（如左图）当 $y_1\neq y_2$ 时，线性限制条件可以写成: $\alpha_1-\alpha_2=k$，根据 k 的正负可以得到不同的上下界，因此统一表示成：

- 下界: $L=\max\left(0, \alpha_2^{\text{old}}-\alpha_1^{\text{old}}\right)$
- 上界: $H=\min\left(C, C+\alpha_2^{\text{old}}-\alpha_1^{\text{old}}\right)$

  （如右图） 当 $y_1=y_2$ 时，限制条件可写成: $\alpha_1+\alpha_2=k$ ，上下界表示成:
- 下界: $L=\max\left(0, \alpha_1^{\text{old}}+\alpha_2^{\text{old}}-C\right)$
- 上界: $H=\min\left(C, \alpha_2^{\text{old}}+\alpha_1^{\text{old}}\right)$

  根据得到的上下界，我们可以得到修剪后的 α_2^{new} ：

$$\alpha_2^{\text{new}}=\begin{cases}H & \alpha_2^{\text{new, unclipped}}>H \\ \alpha_2^{\text{new ,unclipped}} & L \leq \alpha_2^{\text{new ,unclipped}} \leq H \\ L & \alpha_2^{\text{new,unclipped}}<L\end{cases}$$

又 $\alpha_1^{\text{old}} y_1+\alpha_2^{\text{old}} y_2=\alpha_1^{\text{new}} y_1+\alpha_2^{\text{new}} y_2$ ，可得 α_1^{new} ：

$$\alpha_1^{\text{new}}=\alpha_1^{\text{old}}+y_1 y_2\left(\alpha_2^{\text{old}}-\alpha_2^{\text{new}}\right)$$

接下来计算阈值b和差值E_i。

在每次完成两个变量 α_i, α_j 的优化后，都要重新计算阈值 b 。

当 $0<\alpha_1^{\text{new}}<C$ ，由KKT条件可知相应的数据点为支持向量，解得:

$$b_1^{\text{new}}=-E_1-y_1 K_{1,1}\left(\alpha_1^{\text{new}}-\alpha_1^{\text{old}}\right)-y_2 K_{2,1}\left(\alpha_2^{\text{new}}-\alpha_2^{\text{old}}\right)+b^{\text{old}}$$

当$0<\alpha_2^{new}<C$ ，可以得到b_2^{new}:

$$b_2^{\text{new}}=-E_2-y_1 K_{1,2}\left(\alpha_1^{\text{new}}-\alpha_1^{\text{old}}\right)-y_2 K_{2,2}\left(\alpha_2^{\text{new}}-\alpha_2^{\text{old}}\right)+b^{\text{old}}$$

当 α_1^{new} 和 α_2^{new} 同时满足条件时，有 $b^{\text{new}}=b_1^{\text{new}}=b_2^{\text{new}}$ 。 当两个乘子 α_1, α_2 都在边界上，那么b_1^{new}, b_2^{new} 以及它们之间的数都是符合KKT条件的阈值，这时选择它们的中点作为b^{new}:

$$b^{new}=\frac{b_1^{\text{new}}+b_2^{\text{new}}}{2}$$

但通过实践，本次实验采用一种更鲁棒的做法，即对b使用所有支持向量求解的平均值：

$$b=\frac{1}{|S|} \sum_{s \in S}\left(y_s-\sum_{i \in S} \alpha_i y_i \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_s\right)$$

对E_i，取$E_i=sign(g\left(x_i\right))-y_i$，实验发现这样能增加准确率同时减少训练时间。

对 α_i和α_j的选择，因为α_2^{new} 是依赖于 $\left|E_1-E_2\right|$ 的，为了加快计算速度，一种做法是选择 α_2，使其对应的 $\left|E_1-E_2\right|$ 最大。但考虑到计算出最大的$\left|E_1-E_2\right|$较耗时且相对复杂，故在本次实验中对α_i和α_j的选择使用简化方法：对于α_i采用遍历，选定α_i后，对于α_j采用随机选择（除α_i外）。

在实验中，使用矩阵乘法代替for循环，并通过多次判断，减少不必要的计算，从而优化算法效率。

4. 定义损失函数使用梯度下降法

基本想法：最大化间隔的同时，让不满足约束的样本应尽可能少。

$$\min_{\boldsymbol{w}, b} \frac{1}{2}\|\boldsymbol{w}\|^2+C \sum_{i=1}^m \ell_{0/1}\left(y_i\left(\boldsymbol{w}^{\top} x_i+b\right)-1\right)$$

其中 $\ell_{0/1}$ 是 $0/1$ 损失函数。

$$\ell_{0/1}(z)=\begin{cases}1, z<0\\0,\ \text{otherwise}\end{cases}$$

存在的问题：0/1损失函数非凸、非连续，不易优化！

把损失函数换为Hinge Loss：

$$Cost=||\boldsymbol{w}||^2+C\sum_{i=1}^{m}\max\left(0,1-y_i\left(\boldsymbol{w}^{\top}x_i+b\right)\right)$$

接下来使用梯度下降法更新参数。

$$\begin{cases}\frac{dJ}{dw}=2w,\ \ \frac{dJ}{db}=0 & y_i\cdot(wx_i+b)\geq 1\\ \frac{dJ}{dw}=2w-C\mathrm{y_i x_i},\frac{dJ}{db}=-Cy_i & y_i\cdot(wx_i+b)<1:\end{cases}$$

$$w=w-lr\frac{\partial J}{\partial w}$$
$$b=b-lr\frac{\partial J}{\partial b}$$

实验中使用矩阵乘法代替for循环来优化算法效率。

四、实验结果与比较

1. 实验结果

使用五折交叉验证。

- 样本1

  维度：10；

  样本数：5000；

  错标率：0.04。

  - SMO

    参数：dim = 10, max_iter=100, C=1, tol = 1e-8, epsilon = 1e-8

轮数	Accuracy	time（s）
1	0.944	18.901264000000083
2	0.951	18.753948799989303
3	0.956	19.18545720000111
4	0.948	20.798065799986944
5	0.945	20.169730000008713
average	0.9488	19.56169315999723

  - Gradient Descent

    参数：dim = 10, lr=0.01, max_iter=100, C = 1

轮数	Accuracy	time（s）
1	0.945	0.012744399995426647
2	0.954	0.0126419000007445
3	0.957	0.012590400001499802
4	0.949	0.014343699993332848
5	0.945	0.009860899997875094
average	0.95	0.012436259997775779

- sklearn

  参数：penalty = 'l2', loss='hinge', max_iter=100000。其他均为默认。

轮数	Accuracy	time（s）
1	0.945	0.2769842000125209
2	0.954	0.2531458000012208
3	0.957	0.2692286000092281
4	0.949	0.26876520000223536
5	0.945	0.23756069999944884
average	0.95	0.2611369000049308

- 样本2

  维度：20；

  样本数：10000；

  错标率：0.0382。

  - SMO

    参数：dim = 20, max_iter=100, C=1,tol = 1e-8, epsilon = 1e-8

轮数	Accuracy	time（s）
1	0.9515	43.60137220000615
2	0.948	47.81860759999836
3	0.947	51.908864600001834
4	0.941	54.914126599993324
5	0.9585	59.3508620000066
average	0.9492	51.51876660000126

  - Gradient Descent

    参数：dim = 20, lr=0.01, max_iter=100, C = 1

轮数	Accuracy	time（s）
1	0.943	0.027651399999740534
2	0.9495	0.02711049999925308
3	0.957	0.024727700001676567
4	0.949	0.026623400000971742
5	0.9555	0.026343600009568036
average	0.9508	0.02649132000224199

- sklearn

  参数：penalty = 'l2', loss='hinge', max_iter=100000。其他均为默认。

轮数	Accuracy	time（s）
1	0.943	1.4466724000085378
2	0.9495	1.4882093999913195
3	0.957	1.46267780000926
4	0.949	1.4921311999933096
5	0.9555	1.4843074999953387
average	0.9508	1.4747996599995532

- 样本3

  维度：30；

  样本数：15000；

  错标率：0.037。

  - SMO

    参数：dim = 30, max_iter=100, C=1,tol = 1e-8, epsilon = 1e-8

轮数	Accuracy	time（s）
1	0.9537	192.71647959999973
2	0.954	220.26522800000384
3	0.9537	221.9483368999936
4	0.9533	227.10284779999347
5	0.9507	220.8343739999982
average	0.9531	216.57345325999776

  - Gradient Descent

    参数：dim = 30, lr=0.01, max_iter=100, C = 1

轮数	Accuracy	time（s）
1	0.956	0.0462198999885004
2	0.9513	0.046176400006515905
3	0.95	0.050320300011662766
4	0.9513	0.057560899993404746
5	0.9553	0.046223699988331646
average	0.9528	0.04930023999768309

- sklearn

  参数：penalty = 'l2', loss='hinge', max_iter=100000。其他均为默认。

轮数	Accuracy	time（s）
1	0.956	3.7866291999816895
2	0.9513	3.889095799997449
3	0.95	3.8688754999893717
4	0.9513	3.8002525999909267
5	0.9553	3.840269000007538
average	0.9528	3.837024419993395

3. 方法比较

SMO 与 Grandient Descent 在 max_iter > 10 以后准确率增长不大，sklearn.svm 中的 linearSVC 在 max_iter > 10000 后才开始具有高准确率。分别取三者准确率相对较高时对应的 max_iter 与其他指标，进行比较。

(dim,num)	评价指标	SMO	Gradient Descent	sklearn. svm.linearSVC
	max_iter	100	100	100000
(10,1000)	accuracy	0.9488	0.95	0.95
	time(s)	19.5617	0.0124	0.2611
	max_iter	100	100	100000
(20,10000)	accuracy	0.9492	0.9508	0.9508
	time(s)	51.5188	0.0265	1.4748
	max_iter	100	100	100000
(30,15000)	accuracy	0.9531	0.9528	0.9528
	time(s)	179.9977	0.0493	3.8370

（1）准确率：

三种方法得到的准确率相差不大。在数据规模较小时，Gradient Descent 与 sklearn. svm.linearSVC 略高；在数据规模较大时，三者差异减小，甚至SMO略高。

（2）收敛速度：

SMO 与 Gradient Descent 在迭代一轮时准确率就大于0.92，收敛速度较快，sklearn. svm.linearSVC 收敛速度较慢。

（3）耗时：

单位迭代次数的耗时为 sklearn. svm.linearSVC < Gradient Descent < SMO 。

但 Gradient Descent 达到最优准确率的耗时最短。