ML-lab2-report

1. 实验原理

†1.1 线性可分**SVM**算法过程

input: 线性可分的m个样本$\{(x_i, y_i)\}_{i=1}^m, y_i = \{1, -1\}$.

output: 超平面参数ω^*, b^* 和分类决策函数$f(x) = \text{sign}\left(\omega^{*\text{T}} x + b^*\right)$.

算法过程如下:

1. 构造约束优化问题:

$$\begin{aligned} &\min_\alpha \frac{1}{2} \sum_{i=1,j=1}^m \alpha_i \alpha_j y_i y_j x_i^\text{T} x_j - \sum_{i=1}^m \alpha_i \\ &\text{s. t.} \sum_{i=1}^m \alpha_i y_i = 0, 0 \le \alpha_i \le C \end{aligned}$$

2. 用2种算法求出最优的α^*;
3. 计算$\omega^* = \sum_{i=1}^m \alpha_i^* y_i x_i$;
4. 找出所有的支持向量(x_s, y_s), 通过$y_s \left(\sum_{i \in S} \alpha_i y_i x_i^\text{T} x_s + b\right) = 1$计算出每个支持向量对应的$b_s$, 取平均值即得最终的$b^* = \frac{1}{|S|} \sum_{i \in S} b_s$.

可见问题关键在于1.优化问题的求解. 采用两种方法, 梯度下降法和SMO. 下面简要阐述两种算法的步骤:

†1.2 SMO算法

核心步骤如下:

1. 选择第一个变量α_1. 这个变量需要选择在训练集中违反KKT条件最严重的样本点;
2. 选择第二个变量α_2. 选择标准是让$\left|E_1-E_2\right|$有足够大的变化(实际编程中我随机选择α_2, 详见†3.1);
3. 根据α_1, α_2, 求出新的未截断$\tilde{\alpha}_2^{\text{new}}=\alpha_2^k+\frac{y_2\left(E_1-E_2\right)}{K_{11}+K_{22}-2K_{12}}$.
4. 对变量进行截断, 得到α_2^{new}. 截断规则如下:

$$\alpha_2^{\text{new}}=\begin{cases}H, & \tilde{\alpha}_2^{\text{new}}>H \\ \tilde{\alpha}_2^{\text{new}}, & L \leqslant \tilde{\alpha}_2^{\text{new}} \leqslant H \\ L, & \tilde{\alpha}_2^{\text{new}}<L\end{cases}$$

   H, L的求解这里不列出. 需要分类讨论.

5. 利用α_1^{new}和α_2^{new}的关系求出$\alpha_1^{\text{new}}=\alpha_1^{\text{old}}+y_1y_2\left(\alpha_2^{\text{old}}-\alpha_2^{\text{new}}\right)$;
6. 计算b^{k+1}(可参考<统计学习方法>)和更新E_i.

†1.3 梯度下降法

目标函数:

$$\min_{\boldsymbol{w},b}\frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=1}^m\max\left(0,1-y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b\right)\right)$$

计算偏导数进行梯度下降:

$$\frac{\partial\hat{L}\left(w,b,x_i,y_i\right)}{\partial w}=w+\begin{cases}0 & y_i\left(w\cdot x_i+b\right)\geqslant 1 \\ -Cy_ix_i & y_i\left(w\cdot x_i+b\right)<1\end{cases}$$

$$\frac{\partial\hat{L}\left(w,b,x_i,y_i\right)}{\partial b}=\begin{cases}0 & y_i\left(w\cdot x_i+b\right)\geqslant 1 \\ -Cy_i & y_i\left(w\cdot x_i+b\right)<1\end{cases}$$

2. 实验结果分析

†2.1 算法正确性测试

(注:我的错标率是**0.0398,** 我的数据生成维度为**20,** 数量为**10000.** 在**†2.2**中我会用更多样的数据生成方式.)

我的两种方法与sklearn (linear kernel)的对比表格如下:

	准确率	运行时间
SMO(我的)	**91.95%**	**3.58s**
GD(我的)	76%	7.7s
sklearn (`max_iter=4`)	62.95%	0.007s
sklearn (默认设置)	95.8%	25.48s

注: SMO参数设置为

`C=5, toler=0, max_iter=3, random_select_alpha_j=True, delta=1e-5`,

(`random_select_alpha_j`意思为随机选择第二个变量α_2, 详见†3.1.).

GD的参数设置为: `lr=0.001, max_iter=1e5`.

SMO算法中, 我实际进行了3轮iteration, 在3.58s的时间里达到91.95%的准确率, 证明了模型的正确性.

准确率: 91.95% (SMO, our method) v.s. 95.8% (sklearn)
运行时间: 3.58s (SMO, our method) v.s. 25.48s (sklearn).

我的运行时间快很多.我使用的`max_iter`为3, 如果将sklearn的`max_iter`也设置为3, 则sklearn的(准确率,运行时间)=(62.95%,0.007). 这种情况下我的准确率又高非常多.

GD算法中, 在7.7s的时间里达到76%的准确率, 这个方法在我这里就不如SMO. 当然也是因为我没有加更多的东西(只写了最基本的GD). 运行时间和准确率都比我的SMO方法要差很多.

†2.2 相关超参数分析

主要内容: 软间隔`toler`, 数据量, 数据维度.

注: 除`toler`的分析外, SVM_1(SMO)的参数统一为: `C=5, toler=0, max_iter=3`, SVM_2(GD)的参数统一为: `lr=0.001, max_iter=1e5`.

1. 数据量: 如下图所示.

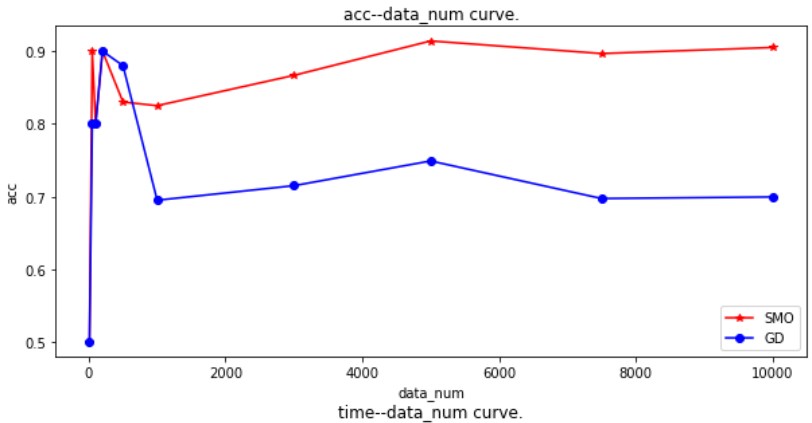
   首先固定数据维度为15, 探究准确率, 时间随数据量在[10, 50, 100, 200, 500, 1000, 3000, 5000, 7500, 10000]的变化. 具体数据见下方的表格. 我绘制了`准确率-数据量`图线和`时间-数据量`图线, . 红线是SMO方法, 蓝线是GD. 可以看出, 随着数据量增长:

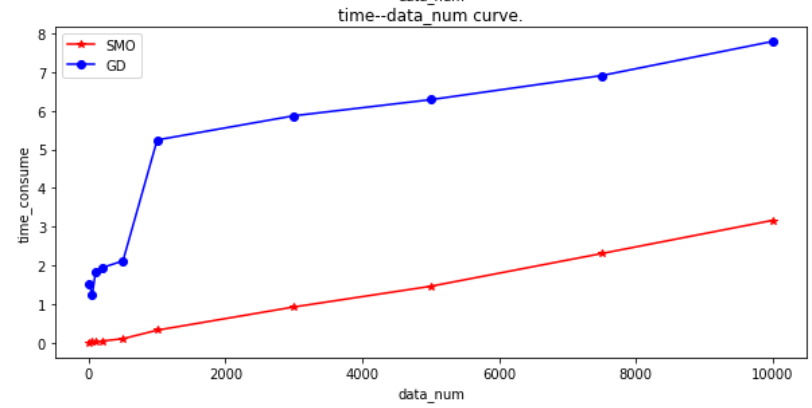
   时间损耗上, num $<$ 1000时, GD的增长速度要更快; 而后SMO, GD的增长速度持平.

   在准确率上, 数量较小(num $<$ 50)时, 两者准确率都有明显提升(这是因为我的维度设置为15, 数据量过少会collapse, 这是变量控制的问题.); 而后维持在一定水平: SMO稳定在0.9, GD稳定在0.7.

exp1

data_num		10	50	100	200	500	1000	3000	5000	7500	10000
time(s)	SMO	0.004	0.013	0.027	0.044	0.102	0.323	0.923	1.458	2.305	3.162
	GD	1.504	1.230	1.841	1.941	2.116	5.242	5.867	6.283	6.906	7.789
acc(%)	SMO	50.0%	90.0%	80.0%	90.0%	83.0%	82.5%	86.7%	**91.4%**	89.7%	90.5%
	GD	50.0%	80.0%	80.0%	**90.0%**	88.0%	69.5%	71.5%	74.9%	69.7%	70.0%

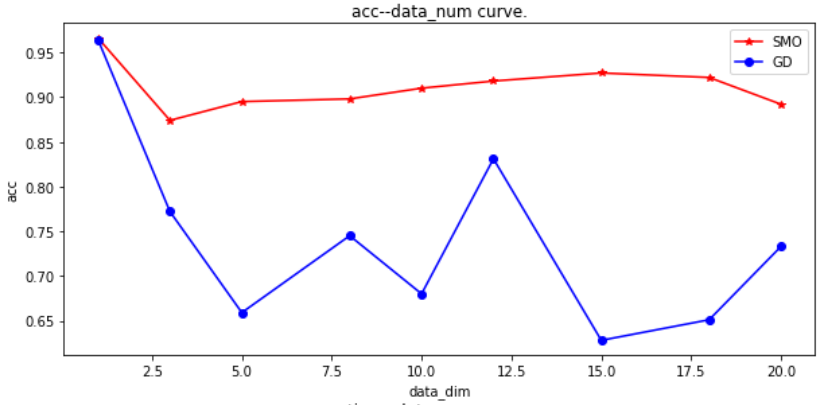


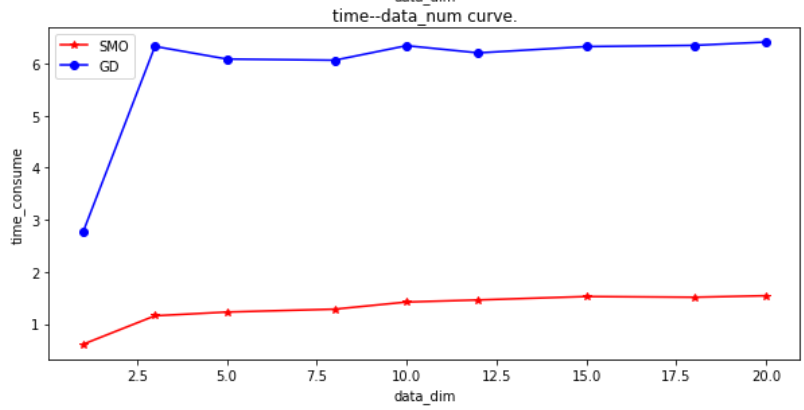


2. 数据维度. 图表如下所示. (注意这里的title有误, 应为`data_dim`). 在固定数目为5000的情况下, 维度设置为[1, 3, 5, 8, 10, 12, 15, 18, 20]. 可以看到: 随着维度的升高, 两个算法的performance都有呈下降的趋势. SMO算法降至0.9左右, GD算法降至均值为0.7, 但是不稳定; 时间上呈现的规律和1.类似.

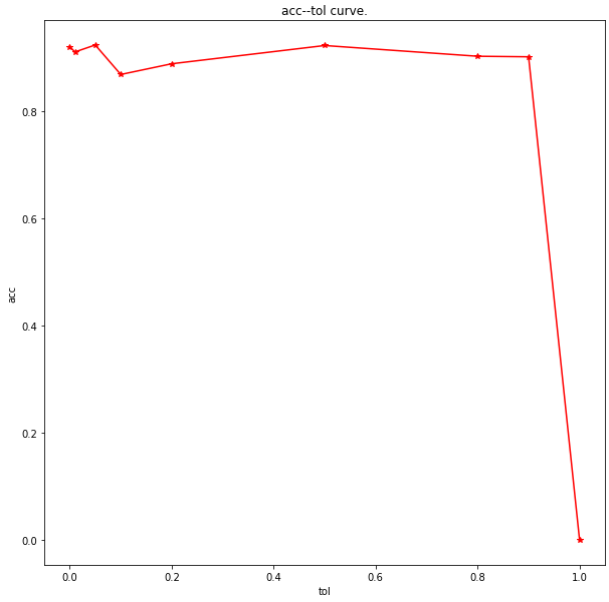
1,2 综合考虑而言, SMO的稳定性, 效果, 时间, 均优于GD.

exp2										
data_dim		1	3	5	8	10	12	15	18	20
time(s)	SMO	0.609	1.156	1.228	1.283	1.420	1.460	1.526	1.512	1.541
	GD	2.764	6.326	6.083	6.060	6.340	6.201	6.322	6.345	6.409
acc(%)	SMO	**96.6%**	87.4%	89.5%	89.8%	91.0%	91.8%	92.7%	92.2%	89.2%
	GD	**96.4%**	77.2%	65.9%	74.5%	68.0%	83.1%	62.8%	65.1%	73.3%





3. 软间隔: 我又探究了SOM中设置的软间隔`toler`对performance的影响, 当`toler=1`时迅速collapse, `<1`时对performance影响不大. 数据生成设置为: `dim=15, num=5000`.



3. 相关编程细节

关于SMO算法的细节比较多, 我的精力主要都集中在了SMO的编程实现上. 基本都写在注释里, 有几个点单独在报告里说明一下.

†3.1 第二个变量α_2的选择问题

算法中的让$|E_1 - E_2|$变化最大的方法, 以及随机选择的方法都尝试过. 但是最终采用随机选择α_2. 算E_2并找$\max|E_1 - E_2|$ 的方法非常耗时. 而随机选择所耗时间很少, 并且准确率也很高(通过上面的图可以看出, 可以稳定在0.9左右). 因此最后设置了SVM1 model的一个属性`random_select_alpha_j`并赋值默认值`True`. (代码注释中的"检测区域"就是当时在排查超时的模块)

†3.2 关于内外循环的问题

SMO算法中, 通常称选择第一个变量α_1为外层循环, 选择α_2 为内层循环. 但是在编程中, 直接在内层循环进行KKT判断, 这样外层循环就只需遍历即可.

我在博客中看到了一种交替遍历的方法, 即在所有数据集上进行单遍扫描 & 在非边界α(不等于边界0或C的α值)中实现单遍扫描, 两者交替进行. 我试了一下, 没觉得有太大的变化. 所以就直接用的全体α单遍扫描方法.

†3.3 关于KKT条件

3.1提到的耗时问题, 一开始并没有马上定位到是选择的原因, 因此尝试了一些可能减少耗时的方法. 比如转换KKT条件:

$$\begin{aligned} y_i g(x_i) > 1 + \xi \ &\Rightarrow y_i \left(g(x_i) - y_i\right) + 1 > 1 + \xi \\ &\Rightarrow y_i E_i > \xi \end{aligned}$$

考虑到我们主要存储E_i, 因此把g转化为E更加直接, 减少运算量. (实际上, 这并不是耗时大的主要原因, 远小于3.1提到的时间损耗)

†3.4 关于参数更新

`update_para`函数中, 需要多次判断相关条件以决定是否对有关参数进行更新.

- 违背了KKT条件, 则正常进行α、E_k、b的更新;
- 截断边界$L = H$, 则不必更新;

- $\eta = K_{11} + K_{22} - 2K_{12} = \left\| \Phi\left(x_1\right) - \Phi\left(x_2\right) \right\|^2$. 是完全平方式. 如果等于0, 则不必继续更新;
- α_2如果没有足够的变化, 则不必继续更新α_1.