序贯最小优化的改进算法*

李建民+, 张 钹, 林福宗

(清华大学 计算机科学与技术系,北京 100084)
(清华大学 智能技术与系统国家重点实验室,北京 100084)

An Improvement Algorithm to Sequential Minimal Optimization

LI Jian-Min+, ZHANG Bo, LIN Fu-Zong

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)
(State Key Laboratory of Intelligent Technology and Systems, Tsinghua University, Beijing 100084, China)
+ Corresponding author: Phn: 86-10-62782266 ext 8426, E-mail: ljm@s1000e.cs.tsinghua.edu.cn
http://www.cs.tsinghua.edu.cn
Received 2002-01-07; Accepted 2002-08-13

Li JM, Zhang B, Lin FZ. An improvement algorithm to sequential minimal optimization. *Journal of Software*, 2003,14(5):918~924.
http://www.jos.org.cn/1000-9825/14/918.htm

Abstract: At present sequential minimal optimization (SMO) algorithm is a quite efficient method for training large-scale support vector machines (SVM). However, the feasible direction strategy for selecting working sets may degrade the performance of the kernel cache maintained in SMO. After an interpretation of SMO as the feasible direction method in the traditional optimization theory, a novel strategy for selecting working sets applied in SMO is presented. Based on the original feasible direction selection strategy, the new method takes both reduction of the object function and computational cost related to the selected working set into consideration in order to improve the efficiency of the kernel cache. It is shown in the experiments on the well-known data sets that computation of the kernel function and training time is reduced greatly, especially for the problems with many samples, support vectors and non-bound support vectors.
Key words: machine learning; support vector machine; sequential minimal optimization; cache

摘 要: 序贯最小优化(sequential minimal optimization,简称 SMO)算法是目前解决大量数据下支持向量机(support vector machine,简称 SVM)训练问题的一种十分有效的方法,但是确定工作集的可行方向策略会降低缓存的效率.给出了 SMO 的一种可行方向法的解释,进而提出了一种收益代价平衡的工作集选择方法,综合考虑与工作集相关的目标函数的下降量和计算代价,以提高缓存的效率.实验结果表明,该方法可以提高 SMO 算法的性能,缩短 SVM 分类器的训练时间,特别适用于样本较多、支持向量较多、非有界支持向量较多的情况.
关键词: 机器学习;支持向量机;序贯最小优化;缓存

* Supported by the National Natural Science Foundation of China under Grant No.60135010 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.G1998030509 (国家重点基础研究发展规划(973))

第一作者简介: 李建民(1971－),男,山东微山人,博士生,主要研究领域为文语转换,机器学习.

© 中国科学院软件研究所 http://www.jos.org.cn

中图法分类号: TP181 文献标识码: A

支持向量机(support vector machines,简称 SVM)[1]以统计学习理论为基础,具有简洁的数学形式、直观的几何解释和良好的泛化能力,是一种解决分类、回归、概率密度估计等问题的有力工具,近年来在手写数字识别、目标识别、文本分类、时间序列预测等许多实际应用中取得了成功.

给定输入空间中的 l 个训练样本$(\boldsymbol{x}_i,y_i)$,$\boldsymbol{x}_i\in R^n$,$y_i\in\{-1,1\}$,$i=1,\ldots,l$,某个核函数(kernel function)$K(\boldsymbol{z}_i,\boldsymbol{z}_j)$和正则化参数 C,训练一个 SVM 二分类器,实质上是求解二次规划(quadratic programming,简称 QP)问题,见式(1).

$$\left.\begin{aligned}&\min_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha})=\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{Q}\boldsymbol{\alpha}-\boldsymbol{e}^T\boldsymbol{\alpha}\\&\text{s.t.}\quad \boldsymbol{y}^T\boldsymbol{\alpha}=0,\ 0\le\alpha_i\le C,\ i=1,\ldots,l\end{aligned}\right\},\tag{1}$$

其中,Hessian 矩阵 $\boldsymbol{Q}$ 是一个半正定矩阵,$\boldsymbol{Q}_{ij}=y_iy_jK(\boldsymbol{x}_i,\boldsymbol{x}_j)$;$\boldsymbol{y}=(y_1,y_2,\ldots,y_l)^T$;$\boldsymbol{e}=(1,1,\ldots,1)^T$.

传统的优化方法,如内点算法、既约梯度法等,在每次迭代中需要利用整个 Hessian 矩阵来更新$\boldsymbol{\alpha}$.而 $\boldsymbol{Q}$ 是一个 $l\times l$ 的非稀疏阵.受普通计算机内存容量的限制,经典的二次规划算法根本无法处理大量数据的问题.因此,$\boldsymbol{Q}$ 的存储和计算成为大规模 SVM 训练问题的瓶颈,制约了 SVM 的应用.

分解算法是解决大量样本下 SVM 训练问题的一类有效方法.它将式(1)分解为一系列规模较小的 QP 问题.在每次迭代中利用传统优化算法求解一个子 QP 问题,更新$\boldsymbol{\alpha}$的一个分量子集(即工作集).各种分解算法的区别在于工作集的大小和生成原则不同.其中,序贯最小优化(sequential minimal optimization,简称 SMO)算法经过不断的改进,成为目前较为有效的 SVM 训练方法.当前最流行的一些 SVM 的训练软件都采用结合可行方向策略的 SMO 算法,并实现了 Kernel Cache 和 shrinking 两种加速方法,比如 LIBSVM[2],SVMTorch[3].利用可行方向策略的 SMO 算法,每次迭代以对 Karush-Kuhn-Tucker(KKT)条件破坏最多的两个 Lagrange 乘子为工作集.但是,这可能使缓存中元素频繁更新,命中率下降,进而导致核函数计算次数增加,训练时间增大.

本文给出 SMO 的一种可行方向法解释,根据这种解释,可直接求出任意工作集带来的目标函数下降值,而不必求出工作集变量的新值.然后提出一种收益代价平衡的工作集选择方法,综合考虑与工作集相关的目标函数下降量和计算代价,提高了缓存的利用效率,缩短了训练时间.本文第 1 节概述 SMO 算法.第 2 节给出 SMO 的可行方向法解释.第 3 节介绍收益代价平衡的工作集选择方法.第 4 节给出实验和分析.最后是结论.

1 SMO 及其改进算法

Platt 首先提出 SMO 算法[4],将工作集大小 q 限定为 2,以得到子 QP 问题的解析解,避免通用优化软件的额外开销.算法采用经验方法而不是文献[5]中的可行方向策略,通过内、外层循环分别确定工作集的两个乘子.同时,利用数据的稀疏性优化核函数的计算,并对线性 SVM 做特殊处理.随后引入 Kernel Cache 和 shrinking 方法[6].

其后,SMO 算法得到了各种改进.Flake 等人[7]针对 SVM 回归问题,改进了经验方法,以提高 Cache 的利用效率.Keerthi 等人[8]修正了优化条件,并针对经验方法提出两个改进措施,以保证算法收敛,并减少迭代次数.随后,Keerthi 等人[9]提出了 generalized SMO (GSMO)算法,利用 violating pair 的概念确定工作集,并指出前面两种改进都是 GSMO 的特例.Chang 等人[2]继而指出文献[8]中的改进 2 是可行方向策略当 q=2 时的特例.

Keerthi 等人[9]证明,$\forall\tau>0$,以 τ-violating pair 为工作集,则 GSMO 算法有限终止,得到式(1)的 τ-近似优化解.Lin[10]证明,在一定条件下,采用可行方向策略的分解算法,其$\{\boldsymbol{\alpha}^k\}$的任意聚点是式(1)的全局最优解.当 q=2 时,Lin[11]证明,无须该条件.Lin 还证明[12],在一定条件下,采用可行方向策略的分解算法具有线性收敛速度.

虽然采用可行方向策略的 SMO 的收敛速度是线性的,但是各子 QP 问题的求解异常简单,所以相对于其他一些算法,该算法还是比较快的,因为训练 SVM 分类器的时间取决于迭代的次数和每次迭代所需要的时间.

© 中国科学院软件研究所 http://www.jos.org.cn

2 SMO 的可行方向法的解释

可行方向法是求解约束非线性规划问题的一类方法.其基本策略是,从一个可行点出发,沿着某可行下降方向进行搜索,得到新的可行点.不同的可行下降方向的生成原则和沿该方向的搜索方式,形成不同的可行方向法.

分解算法与可行方向法具有相似的迭代结构,工作集也对应着一个可行下降方向.但在分解算法中并不沿着该方向进行线搜索,而是求解工作集上的子 QP 问题.因此一般而言,二者是不同的.但当 q=2 时,分解算法等同于可行方向法.利用 SMO 的这种解释,不仅可以简化子 QP 问题的求解,而且可以直接计算目标函数的下降值.

2.1 可行方向法的解释

设采用文献[2]中的可行方向策略确定的工作集为($\boldsymbol{\alpha}_{i1},\boldsymbol{\alpha}_{i2}$),则容易证明,求解相应的二阶子 QP 问题并更新 $\boldsymbol{\alpha}$(即更新 $\boldsymbol{\alpha}$ 的 $i1$ 和 $i2$ 分量),等价于在方向 $\boldsymbol{d}$ 上做一维可行点精确搜索并更新 $\boldsymbol{\alpha}$(即 $\boldsymbol{\alpha}=\boldsymbol{\alpha}^{\text{old}}+a\boldsymbol{d}$),即求解式(2).

$$\left.\begin{aligned}&\min_a U(a)=W(\boldsymbol{\alpha})=W(\boldsymbol{\alpha}^{\text{old}}+a\boldsymbol{d})\\&\text{s.t.}\quad a>0\\&\qquad 0\le\boldsymbol{\alpha}^{\text{old}}+a\boldsymbol{d}\le C\\&\qquad \boldsymbol{y}^T(\boldsymbol{\alpha}^{\text{old}}+a\boldsymbol{d})=0\end{aligned}\right\},\tag{2}$$

其中,$\boldsymbol{\alpha}^{\text{old}}$ 是当前可行点(即 $0\le\boldsymbol{\alpha}^{\text{old}}\le C,\boldsymbol{y}^T\boldsymbol{\alpha}^{\text{old}}=0$);$\boldsymbol{d}$ 是由工作集构造的一个可行下降方向(即 $\boldsymbol{d}_{i1}=y_{i1},\boldsymbol{d}_{i2}=-y_{i2},\boldsymbol{d}_i=0$, $i=1,\ldots,l,i\ne i1,i\ne i2$).容易得到式(2)的解为

$$a=\min(a^*,a\_up),\tag{3}$$

其中,a^*是 $U(a)$的驻点,a_up 是 a 的上界,即

$$\left.\begin{aligned}&a^*=\begin{cases}-\dfrac{\boldsymbol{d}^T(\boldsymbol{Q}\boldsymbol{\alpha}^{\text{old}}-\boldsymbol{e})}{\boldsymbol{d}^T\boldsymbol{Q}\boldsymbol{d}}, & \boldsymbol{d}^T\boldsymbol{Q}\boldsymbol{d}\ne 0\\ +\infty, & \boldsymbol{d}^T\boldsymbol{Q}\boldsymbol{d}=0\end{cases}\\&a\_up=\min(a\_up_{i1},a\_up_{i2})\\&a\_up_j=\begin{cases}C-\boldsymbol{\alpha}_j^{\text{old}}, & \boldsymbol{d}_j=1\\ \boldsymbol{\alpha}_j^{\text{old}}, & \boldsymbol{d}_j=-1\end{cases},\ j=i1,i2\end{aligned}\right\}.\tag{4}$$

因此,可以从可行方向法的角度来理解 SMO 算法:选择工作集($\boldsymbol{\alpha}_{i1},\boldsymbol{\alpha}_{i2}$),即寻找可行下降方向 $\boldsymbol{d}$;求解工作集上的子 QP 问题,即沿 $\boldsymbol{d}$ 进行可行点的精确搜索,确定步长因子 a>0,并更新 $\boldsymbol{\alpha}$.

2.2 目标函数的下降值

在第 k 次迭代中,经过可行点的精确搜索,式(1)的目标函数的下降值为

$$\begin{aligned}\Delta W^k&=W(\boldsymbol{\alpha}^{k+1})-W(\boldsymbol{\alpha}^k)=W(\boldsymbol{\alpha}^k+a^k\boldsymbol{d}^k)-W(\boldsymbol{\alpha}^k)=(\boldsymbol{d}^k)^T\nabla W(\boldsymbol{\alpha}^k)a^k+\frac{1}{2}(\boldsymbol{d}^k)^T\nabla^2W(\boldsymbol{\alpha}^k)\boldsymbol{d}^k(a^k)^2\\&=(\boldsymbol{d}_{i1}^k\quad\boldsymbol{d}_{i2}^k)\begin{pmatrix}\nabla\boldsymbol{W}_{i1}^k(\boldsymbol{\alpha}^k)\\ \nabla\boldsymbol{W}_{i2}^k(\boldsymbol{\alpha}^k)\end{pmatrix}a^k+\frac{1}{2}(\boldsymbol{d}_{i1}^k\quad\boldsymbol{d}_{i2}^k)\begin{pmatrix}\boldsymbol{Q}_{i1i1}&\boldsymbol{Q}_{i1i2}\\ \boldsymbol{Q}_{i1i2}&\boldsymbol{Q}_{i2i2}\end{pmatrix}\begin{pmatrix}\boldsymbol{d}_{i1}^k\\ \boldsymbol{d}_{i2}^k\end{pmatrix}(a^k)^2\\&=(\boldsymbol{d}_{\text{sub}}^k)^T\nabla\boldsymbol{W}_{\text{sub}}^ka^k+\frac{1}{2}(\boldsymbol{d}_{\text{sub}}^k)^T\boldsymbol{Q}_{\text{sub}}^k\boldsymbol{d}_{\text{sub}}^k(a^k)^2.\end{aligned},$$

其中,a^k是第 k 次迭代中可行点精确搜索得到的步长,即式(2)的最优解.为书写简洁,下面略去迭代记号上标 k.

可以注意到,式(2)的目标函数与 $\Delta W(a)=W(\boldsymbol{\alpha}+a\boldsymbol{d})-W(\boldsymbol{\alpha})$ 只相差一个常数,因此对于给定的工作集($\boldsymbol{\alpha}_{i1}$, $\boldsymbol{\alpha}_{i2}$),可以通过式(5)确定其带来的目标函数的最大下降值:

© 中国科学院软件研究所 http://www.jos.org.cn

$$\left.\begin{aligned}&\min_a \Delta W(a)=(\boldsymbol{d}_{\text{sub}})^T\nabla \boldsymbol{W}_{\text{sub}}a+\frac{1}{2}(\boldsymbol{d}_{\text{sub}})^T\boldsymbol{Q}_{\text{sub}}\boldsymbol{d}_{\text{sub}}a^2\\ &\text{s.t.}\quad a>0\\ &\qquad 0\le \boldsymbol{\alpha}^{\text{old}}+a\boldsymbol{d}\le C\\ &\qquad \boldsymbol{y}^T(\boldsymbol{\alpha}^{\text{old}}+a\boldsymbol{d})=0\end{aligned}\right\}. \tag{5}$$

在可行方向策略中,需要利用$\nabla \boldsymbol{W}(\boldsymbol{\alpha})=\boldsymbol{Q}\boldsymbol{\alpha}-\boldsymbol{e}$确定工作集,所以,在算法实现中保存$\nabla \boldsymbol{W}(\boldsymbol{\alpha})$,在每次迭代中加以更新.而$\boldsymbol{Q}$的对角元素$\boldsymbol{Q}_{ii}$,$i$=1,…,$l$,在子QP的求解中频繁出现,可以在算法实现中提前计算并保存元素$\boldsymbol{Q}_{ii}$.因此,一旦确定了工作集,$\boldsymbol{d}_{\text{sub}}$,$\nabla \boldsymbol{W}_{\text{sub}}$均已知,只有当行$\boldsymbol{Q}_{i1}$和$\boldsymbol{Q}_{i2}$不在缓存中时,$\boldsymbol{Q}_{i1i2}$是未知的.所以,最多只需计算一次核函数,并求解式(5),即可得到式(1)的目标函数的下降值.式(5)为一元二次函数在区间上的极值问题,其解为

$$\Delta W=\begin{cases}\dfrac{1}{2}(\boldsymbol{d}_{\text{sub}})^T\nabla \boldsymbol{W}_{\text{sub}}a^*, & a^*\le a\_up\\ (\boldsymbol{d}_{\text{sub}})^T\nabla \boldsymbol{W}_{\text{sub}}a\_up+\dfrac{1}{2}(\boldsymbol{d}_{\text{sub}})^T\boldsymbol{Q}_{\text{sub}}\boldsymbol{d}_{\text{sub}}a\_up^2, & a^*>a\_up\end{cases}, \tag{6}$$

其中a_up,a^*由式(4)给出.

3 收益代价平衡的工作集选择方法

SMO算法收敛较慢,迭代次数较多.在总的计算量中,核函数的计算通常占主要地位.Kernel Cache方法通过缓存$\boldsymbol{Q}$中元素,减少了核函数的重复计算.但是多数SMO中工作集的选择算法,包括Platt的经验、Keerthi的改进和可行方向策略,目的是使目标函数尽可能多地下降.因而算法对缓存的访问很不规则,与常用的Least Recent Used(LRU)的缓存替换原则有冲突.当缓存容量相对于训练问题的规模较小时,就会出现对缓存的频繁更新,命中率下降,重复计算的减少效果就会大打折扣.而且因为维护缓存的开销,训练时间反而有可能增加.

SMORCH[7]在特定情况下关闭缓存,防止过度不规则的访问对缓存的破坏.同时,在经验方法的内层循环中,从对应的$\boldsymbol{Q}$中元素存于缓存中的乘子(以下简称为被缓存的乘子)中,选取使目标函数下降最多的作为工作集中的第2个元素.而在SVM$^{\text{light}}$[5]中,当采用非线性核函数且工作集的大小超过4时,要求工作集中至少有一半乘子被缓存.上述方法的基本思想是优先选取被缓存的乘子,以提高缓存的命中率.但是这可能会造成SMO的效率低下,因为每次迭代中目标函数的下降可能很少,从而导致迭代次数很多.

可行方向策略需要利用梯度$\nabla \boldsymbol{W}(\boldsymbol{\alpha})$选择工作集.因此,在SMO算法的每次迭代中,$(\boldsymbol{\alpha}_{i1},\boldsymbol{\alpha}_{i2})$上的子QP问题求解之后,必须利用$\boldsymbol{Q}$的第$i1$和$i2$行来更新.所以,如果在被缓存的乘子中选择工作集,可以减少$\boldsymbol{Q}$中两行元素的计算;而如果不加限制,则可能获得较大的目标函数下降量.这实际上是代价与收益的折衷问题.

基于这种考虑,我们提出了收益代价平衡的工作集选择方法,力图使每次迭代中的收益和代价在一定程度上达到平衡,兼顾迭代次数与缓存效率.首先在所有的乘子中确定工作集,即求解式(7)[2].

$$\left.\begin{aligned}i1&=\arg\max\left(\left\{-\nabla \boldsymbol{W}(\boldsymbol{\alpha})_t\middle|y_t=1,\alpha_t<C\right\},\left\{\nabla \boldsymbol{W}(\boldsymbol{\alpha})_t\middle|y_t=-1,\alpha_t>0\right\}\right),\\ i2&=\arg\min\left(\left\{\nabla \boldsymbol{W}(\boldsymbol{\alpha})_t\middle|y_t=-1,\alpha_t<C\right\},\left\{-\nabla \boldsymbol{W}(\boldsymbol{\alpha})_t\middle|y_t=1,\alpha_t>0\right\}\right),\end{aligned}\right\} \tag{7}$$

然后在被缓存的乘子中确定工作集,就是求解式(8).

$$\left.\begin{aligned}i1&=\arg\max\left(\left\{-\nabla \boldsymbol{W}(\boldsymbol{\alpha})_t\middle|y_t=1,\alpha_t<C,\boldsymbol{Q}_t\text{ in cache}\right\},\left\{\nabla \boldsymbol{W}(\boldsymbol{\alpha})_t\middle|y_t=-1,\alpha_t>0,\boldsymbol{Q}_t\text{ in cache}\right\}\right),\\ i2&=\arg\min\left(\left\{\nabla \boldsymbol{W}(\boldsymbol{\alpha})_t\middle|y_t=-1,\alpha_t<C,\boldsymbol{Q}_t\text{ in cache}\right\},\left\{-\nabla \boldsymbol{W}(\boldsymbol{\alpha})_t\middle|y_t=1,\alpha_t>0,\boldsymbol{Q}_t\text{ in cache}\right\}\right),\end{aligned}\right\} \tag{8}$$

其中$\boldsymbol{Q}_t$是$\boldsymbol{Q}$的第t行.然后根据两个工作集的收益和代价,确定最终的工作集.

因此,计算一个工作集所带来的目标函数的下降量,成为新算法的关键.幸运的是,SMO算法的工作集仅包含两个Lagrange乘子,使计算变得较为简单.一种方法是,求解关于该工作集的二阶子QP问题,进而得到新的目标函数值和目标函数的下降量.但是,借助SMO的可行方向法的解释,可以直接得到目标函数的下降量.

算法 3.1. 收益代价平衡的工作集选择算法.

Step 1. 求解式(7),在所有的Lagrange乘子中确定一个工作集I_{all}.

© 中国科学院软件研究所 http://www.jos.org.cn

Step 2. 如果 I_{all} 已经满足优化条件,则工作集为空,即不存在可行下降方向,当前$\boldsymbol{\alpha}$为近似最优解;否则转 Step 3.

Step 3. 求解式(8),在被缓存的 Lagrange 乘子中确定一个工作集 I_{cache}.

Step 4. 若 I_{cache} 不存在,或者 I_{cache} 已经满足优化条件,或者 I_{cache} 与 I_{all} 相同,则选择工作集 I_{all};否则转 Step 5.

Step 5. 利用式(6)分别计算 I_{cache} 引起的目标函数的下降值 $R_{cache}=-\Delta W$,I_{all} 引起的目标函数的下降值 $R_{all}=-\Delta W$.比较 R_{all} 和 R_{cache},如果 $R_{cache}\geq coef\cdot R_{all}$,则选择工作集 I_{cache},否则选择工作集 I_{all}.

在上述算法中,参数 *coef* 用来平衡目标函数的下降值和为此付出的计算代价,可近似为每次迭代中以 I_{cache} 和 I_{all} 为工作集的计算量之比再乘上一个系数.可以看到,当 *coef*=0 时,算法即代价优先算法,尽量减少每次迭代中的计算代价;而当 *coef*=∞时,收益优先算法希望获得较大的目标函数下降量,就是一般的可行方向策略.

需要指出,算法 3.1 得到的工作集都是由 violating pair 构成的,根据文献[9],此算法应用于 SMO 是收敛的.

4 实验结果及分析

4.1 实验内容

在 LIBSVM 的基础上,我们实现了收益代价平衡算法.作为对照,还实现了两个算法,一个是随机算法,即按照等概率随机选取 I_{cache} 和 I_{all};另一个是代价优先算法,即优先选取 I_{cache}.所有算法均采用 C++实现,并利用 C++ Builder 5.0 编译.在实现文中算法时,没有采取任何优化措施,以便突出算法本身的区别.

我们在两个公共测试数据库 UCI Adult 和 Web 数据集[4]上对上述 3 个算法,LIBSVM 2.31 和 SVM^{light} 3.50 做了 12 个实验,以比较在不同数量的训练样本、不同数量的支持向量、不同容量的缓存、是否采用 shrinking、不同的参数 *coef* 等情况下的性能.实验的硬件平台为 PIII 600 笔记本,128M RAM,操作系统为 Window 98 SE.SVM 的训练参数以及 LIBSVM 和 SVM^{light} 的参数,都采用 Platt[4]和 Joachims[5]的建议值.

4.2 实验结果

受篇幅限制,这里仅给出部分实验结果,见表 1~表 4.其中,随机方法的数据是两次训练所得数据的平均值.

Table 1 Runtime comparison among these methods on small training sets

表 1 小训练样本集上各种方法的训练时间比较

Experiment	I_{cache} first (s)	Algorithm 3.1 (s)*	Random (s)	LIBSVM (s)	SVM^{light} (s)
Adult-4,40M	115.225	30.325	30.848	28.485	
Adult-4,20M	104.080	31.975	32.346	29.815	
Adult-4,40M,shrinking		30.850	31.528	29.840	26.26
Adult-4,20M,shrinking		31.010	31.270	29.910	30.27
Web-4,40M	610.656	39.740	38.738	26.235	
Web-4,20M		40.605	40.273	28.315	
Web-4,40M,shrinking		24.300	24.336	21.380	39.70
Web-4,20M,shrinking		33.410	37.176	36.434	40.51

Table 2 Runtime comparison among these methods on large training sets

表 2 大训练样本集上各种方法的训练时间比较

Experiment	I_{cache} first (s)	Algorithm 3.1 (s)**	Random (s)	LIBSVM (s)	SVM^{light} (s)
Adult,40M	8 203.825	1 851.535	2 346.365	2 532.712	
Adult,20M	6 363.925	1 938.730	2 674.803	3 326.389	
Adult,40M,shrinking		1 457.050	1 463.452	1 429.419	1 636.19
Adult,20M,shrinking		1 461.475	1 483.279	1 467.235	1 635.20
Web,40M	24 173.472	2 157.105	2 790.098	3 064.085	
Web,20M		2 275.110	2 969.760	3 418.781	
Web,40M,shrinking		1 354.490	1 419.555	1 457.495	1 508.34
Web,20M,shrinking		1 408.975	1 499.485	1 581.069	1 653.53

* *coef* 依次取 0.25,0.25,0.5,1,1,0.5,0.5,0.5.

** *coef* 依次取 0.1,0.1,0.1,0.5,0.5,0.25,0.5,0.5.

© 中国科学院软件研究所 http://www.jos.org.cn

Table 3 Runtime comparison among these methods on moderate training set with different number of non-bound support vectors

表 3 较大训练样本集上不同非有界支持向量数目下各种方法的训练时间比较

Experiment	Algorithm 3.1 (s)***	Random (s)	LIBSVM (s)	SVM^{light} (s)
Adult-7,*C*=1,40M	378.435	384.766	343.000	
Adult-7,*C*=1,40M,shrinking	331.695	335.582	322.175	389.44
Adult-7,*C*=10,40M	1 362.100	2 546.908	3 014.405	
Adult-7,*C*=10,40M,shrinking	717.000	777.498	950.884	1 472.15
Adult-7,*C*=100,40M	6 440.365	18 501.719	25 261.550	
Adult-7,*C*=100,40M,shrinking	2 780.134	4 514.845	5 193.210	9 671.27

Table 4 Comparison of runtime, number of iterations and number of kernel evaluations among these methods on Adult data set with cache size being 40M

表 4 Adult 数据集上各种方法(缓存容量为 40M)的训练时间、迭代次数和核函数计算次数比较

	I_{cache} first (*coef*=0)	Algorithm 3.1 (s)					Random	LIBSVM (*coef*=∞)
		coef=0.05	*coef*=0.075	*coef*=0.1	*coef*=0.25	*coef*=0.5		
Runtime (s)	8 203.825	1 966.221	1 858.985	1 851.535	1 867.105	1 906.630	2 346.365	2 532.712
Number of iterations	406 716	36 951	33 906	30 640	26 595	25 247	24 625	24 621
Number of kernel evaluations	491 165 208	505 752 984	504 743 562	511 353 648	527 178 780	574 263 432	747 135 090	903 725 748

4.3 结果分析

4.3.1 参数 *coef* 对训练时间的调节作用

和预期的一样,在算法 3.1 中,参数 *coef* 对迭代次数和核函数的计算次数具有平衡作用,进而影响总的训练时间.从表 4 中可以看出,随着 *coef* 的增大,迭代次数呈减少趋势,核函数的计算次数呈增多趋势,而总的训练时间先是逐渐减少,然后又逐渐增大,即存在极小点.所以,通过选择适当的 *coef*,可以减少 SVM 的训练时间.最佳 *coef* 与 Kernel Cache 的容量、训练样本的数量和稀疏程度、核函数的复杂程度等因素有关.

实验数据同时表明,在最佳 *coef* 附近,训练时间的变化并不显著.因此,即便不能找到最佳 *coef*,只要在其某个邻域内,都能够有效地减少训练时间.这个性质意味着收益代价平衡的工作集选择算法具有很大的可操作性,只要大致估计出较好的 *coef* 即可.

4.3.2 各种方法比较

同样采用收益优先算法,SVM^{light}(shrinking 模式)的迭代次数和核函数计算次数与 LIBSVM(shrinking 模式)大致相当,但是由于通用优化软件带来一定的额外开销,训练时间比 LIBSVM 长.代价优先方法虽然能减少核函数的计算次数,甚至 1/2 以上,但是由于候选可行下降方向被限制在较小范围内,迭代次数过多,训练时间远远超过其他方法,因而并不适用于 SMO 算法.随机方法在多数情况下不如收益优先方法(以下除非特别声明,收益优先算法是指 LIBSVM),但当样本很多时,则优于收益优先算法,而不如平衡算法取最佳的 *coef*.收益优先方法当样本较少时强于最佳 *coef*,但是当样本较多,特别是非有界支持向量较多时,最佳 *coef* 的训练时间则明显少于收益优先方法.

收益优先和平衡算法存在性能差异的原因在于:在 SVM 的训练过程中,Lagrange 乘子从初值 0 逐步更新成最优解,至少访问一次支持向量对应的 $\boldsymbol{Q}$ 中的行,而访问非有界支持向量对应的行的次数可能较多.当训练样本相对于缓存容量较少时,缓存可以存放较多 $\boldsymbol{Q}$ 的行,命中率相对较高,所以收益优先和平衡算法中核函数计算次数相差很少,但是平衡算法会对缓存中的工作集多做一些迭代,而且有其他一些计算,比如估计目标函数的下降值,所以总的训练时间略多于收益优先算法.而当训练样本相对较多时,缓存存放的行就会变少,收益优先算法可能会用当前工作集对应的 $\boldsymbol{Q}$ 的行替换缓存中已有的行(如果缓存是满的),而不久又重新计算被舍弃的行.这种对缓存的频繁更新导致核函数的计算次数急剧变多,特别是当非有界支持向量较多时.而对于平衡算法,只要

*** *coef* 依次取 0.25,0.1,0.1,0.5,0.25,0.5.

© 中国科学院软件研究所 http://www.jos.org.cn

缓存中的工作集能够使目标函数的下降值超过一定幅度,就会尽量利用被缓存的乘子,因而能够减少核函数的计算次数.

实验数据证明了这一点.对于数据子集 Adult,由于缓存容量不同,平衡算法(取 *coef*=0.1,下面均略去 *coef* 的最佳值)的核函数计算次数比收益优先算法分别下降约 43%和 54%,总的训练时间也分别下降约 27%和 42%.对于数据子集 Adult-7,当 C=10 和 C=100 时,由于非有界支持向量数目的不同,平衡算法中核函数计算次数比收益优先算法分别下降约 79%和 92%,总的训练时间也分别下降约 55%和 75%.

从实验数据中也可以看出,采用 shrinking 方法,逐步估计出有界支持向量和非支持向量,从而避免对其进行优化,以减少核函数的计算次数和训练时间,尤其适合有界支持向量较多的情况.对于数据子集 Adult,收益优先算法+shrinking 的训练时间与平衡算法+shrinking 基本相同,而对于数据子集 Web,平衡算法+shrinking 略好,减少约 10%,这是因为 Web 数据集中的非有界支持向量的数量比 Adult 数据集中的多一些.但是对于 Adult-7,当 C 分别取 10 和 100 时,平衡算法+shrinking 的性能大大优于收益优先算法+shrinking,核函数的计算次数分别下降约 37%和 71%,训练时间分别下降约 25%和 46%,而相对于 SVM^{light},训练时间更是分别下降约 51%和 71%.因为此时非有界支持向量数目较大,比例较高,而 shrinking 方法并不能处理这种情况.

5 结 论

本文首先给出了 SMO 算法的可行方向法的解释,并据此得到任意工作集造成的目标函数下降值的计算公式;然后介绍了一种应用于 SMO 算法的工作集选择方法,在可行方向策略的基础上,考虑了 Kernel Cache 的利用效率,在目标函数的下降幅度和所需要的计算量之间得到了平衡.实验表明,这种方法可以提高 SMO 算法的性能,缩短 SVM 分类器的训练时间,特别适用于样本较多、支持向量较多、非有界支持向量较多的情况.今后我们将把这种方法推广到 SVM 回归.另外,使这种方法与 shrinking 方法更好地结合,也是值得研究的方向.

致谢 台湾大学计算机科学与信息工程系 Chang Chih-Chun 和 Lin Chih-Jen 的训练软件 LIBSVM 为我们的方法和实验提供了一个平台,在此向他们表示感谢.

References:

[1] Burges C. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 1998,2(2):1~43.

[2] Chang CC, Lin, CJ. LIBSVM: A library for support vector machines (Version 2.3). 2001. http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf. 2001.

[3] Collobert R, Bengio S. SVMTorch: A support vector machine for large-scale regression and classification problems. Journal of Machine Learning Research, 2001,1:143~160.

[4] Platt J. Fast training of support vector machines using sequential minimal optimization. In: Schölkopf B, Burges C, Smola A, eds. Advances in Kernel Methods——Support Vector Learning. Cambridge, MA: MIT Press, 1999. 185~208.

[5] Joachims T. Making large-scale support vector machine learning practical. In: Schölkopf B, Burges C, Smola A, eds. Advances in Kernel Methods——Support Vector Learning. Cambridge, MA: MIT Press, 1999. 169~184.

[6] Platt J. Using analytic QP and sparseness to speed training of support vector machines. In: Kearns M, Solla S, Cohn D, eds. Advances in Neural Information Processing Systems 11. Cambridge, MA: MIT Press, 1999. 557~563.

[7] Flake G, Lawrence S. Efficient SVM regression training with SMO. Machine Learning, 2002,46(1/3):271~290.

[8] Keerthi S, Shevade S, Bhattcharyya C, Murthy K. Improvements to Platt's SMO algorithm for SVM classifier design. Neural Computation, 2001,13(3):637~649.

[9] Keerthi S, Gilbert E. Convergence of a generalized SMO algorithm for SVM classifier design. Machine Learning, 2002,46(1/3):351~360.

[10] Lin CJ. On the convergence of the decomposition method for support vector machines. IEEE Transactions on Neural Networks, 2001,12(6):1288~1298.

[11] Lin CJ. Asymptotic convergence of an SMO algorithm without any assumptions. 2001. http://www.csie.ntu.edu.tw/~cjlin/papers/q2conv.pdf.

[12] Lin CJ. Linear convergence of a decomposition method for support vector machines. 2001. http://www.csie.ntu.edu.tw/~cjlin/papers/linearconv.pdf.

© 中国科学院软件研究所 http://www.jos.org.cn