中国科学: 信息科学 2016 年 第 46 卷 第 9 期: 1276–1287
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
论 文

基于分解策略的多目标演化子集选择算法

钱超[①②③], 周志华[①②*]

① 南京大学计算机软件新技术国家重点实验室, 南京 210023
② 软件新技术与产业化协同创新中心, 南京 210023
③ 中国科学技术大学计算机科学与技术学院, 合肥 230027
* 通信作者. E-mail: zhouzh@nju.edu.cn

收稿日期: 2016–03–03; 接受日期: 2016–04–22; 网络出版日期: 2016–09–02
国家自然科学基金 (批准号: 61333014, 61321491) 资助项目

摘要 在许多现实的机器学习任务中, 经常遇到从一组变量中挑选一个子集的问题, 即子集选择问题. 对于这类问题的求解是 NP 难的. 最近, 一种基于多目标演化算法的子集选择算法 POSS 被提出; 无论是在理论上还是在实验上, POSS 方法均获得了目前的最佳性能. 然而, 当问题规模很大的时候, POSS 方法的运行时间变得难以令人满意, 这阻碍了其在大规模实际问题中的应用. 提出了一种基于分解策略的多目标演化子集选择算法 DPOSS. DPOSS 方法将整个子集空间分解成多个子空间, 并依次调用 POSS 方法来求解. 在理论上, DPOSS 方法在获得和 POSS 方法相同近似性能下界的同时, 运行时间随着分解个数的增加超线性下降. 实验结果验证了这一理论, 并显示出, DPOSS 方法的实际性能随着分解个数的增加略有下降, 但依然优于以往的贪婪算法.

关键词 机器学习 子集选择 多目标优化 多目标演化算法 分解策略

1 引言

在许多现实的机器学习任务中, 经常遇到这样一类问题: 从给定的 n 个变量中选择大小不超过 k 的一个子集来优化某个给定的目标. 这类问题被称为子集选择 (subset selection) 问题, 其起源于矩阵列选择问题, 即从一个给定的矩阵中选择若干列以使选择的这些列能够最好地恢复该矩阵 [1]. 子集选择问题现已出现在各种各样的应用中, 例如选择性集成 [2]、属性选择 [3]、稀疏学习 [4] 等.

然而, 对于子集选择问题的求解是 NP 难的 [5]. 已有许多国内外研究者针对该问题提出了多项式时间的近似方法, 主要可以分成两类: 贪婪算法和凸放松方法. 贪婪算法循环地加入或删除一个使当前目标最优化的变量, 直至变量数目达到 k 为止 [6,7]. 尽管在实际应用中被广泛使用, 但是贪婪算法的优化能力往往受其贪婪行为所限, 易于陷入局部最优解. 凸放松方法通常先把对变量子集大小的约束 (也就是 ℓ_0 范数) 替换成凸约束 (比如 ℓ_1 范数 [8] 和 elastic net penalty [9]), 然后再去优化放松后的问题. 这类方法的缺陷是放松后问题的最优解可能和原始问题的最优解差别很大.

引用格式: 钱超, 周志华. 基于分解策略的多目标演化子集选择算法. 中国科学: 信息科学, 2016, 46: 1276–1287, doi: 10.1360/N112016-00045

近年来, 具有理论支持的多目标演化算法被成功应用于选择性集成和最小集合覆盖问题 [10~12]. 受此启发, Qian 等 [13] 提出了一种多目标演化子集选择算法 POSS. POSS 方法的主要过程如下: 首先, 子集选择问题被转化为一个二目标优化问题: 最优化原始目标, 同时最小化变量子集大小; 然后, 采用一种多目标演化算法求解该转换得到的二目标优化问题; 最后, 从求得的解集中挑选一个满足子集大小约束的最好解输出. 值得注意的是, 对于二目标优化问题, 一个解对应的是一个目标向量而不再是一个目标值, 故解与解之间可能是不可比的, 因此多目标演化算法找到的是一个包含互不可比解的集合而不是一个唯一的解. 对于子集选择问题的一个代表实例稀疏回归问题 (sparse regression) [14], POSS 方法被证明可以获得目前已知的最优近似性能; 该性能之前由贪婪算法获得 [15]. 而且, 在稀疏回归问题的一个重要子类 [16] 上, POSS 方法被证明可以找到最优解. 在实验上, POSS 方法的性能也显示出比贪婪算法和凸放松方法显著的优势.

从 Qian 等 [13] 对 POSS 方法的分析可见, 为了获得好的性能, 它需要进行 $2ek^2n$ 次目标函数评估. 因此, 当总的变量数目 n 和子集大小约束 k 很大的时候, POSS 方法的计算开销将变得无法令人满意, 这阻碍了 POSS 方法在大规模实际问题中的应用. 为此, Qian 等 [17] 最近提出了 POSS 方法的一个并行版本, 使其在保持性能不变的同时, 运行时间随着处理器数目的增加而几乎线性下降.

本文提出了一种基于分解策略的多目标演化子集选择算法 DPOSS. POSS 方法直接在整个子集空间中搜索一个最优子集; 而 DPOSS 方法首先将整个子集空间分解成多个子空间, 然后依次调用 POSS 方法来逐个搜索, 并且在前一个子空间中搜索到的最好解将作为下一子空间搜索的初始解. 在理论上, 我们证明了 DPOSS 方法在获得和 POSS 方法相同近似性能下界的同时, 运行时间随着分解个数的增加超线性下降. 实验结果验证了运行时间超线性下降这一理论结果; 并显示出, DPOSS 方法的实际性能随着分解个数的增加略有下降, 但依然比以往获得最佳理论近似性能的贪婪算法要好.

本文后续部分组织如下. 第 2 节介绍子集选择问题. 第 3 节介绍本文提出的 DPOSS 方法. 第 4 和 5 节分别给出理论分析和汇报实验结果. 最后, 第 6 节总结全文.

2 子集选择问题

本节首先介绍子集选择问题, 然后介绍 Qian 等 [13] 提出的多目标演化子集选择算法 POSS.

2.1 问题定义

正如定义 1 所示, 子集选择问题旨在从一个变量集合 V 中选择一个子集 S, 在满足 $|S| \leqslant k$ 的约束条件下, 使给定目标函数最小化. 本文用 $|\cdot|$ 表示一个集合的大小, 即所包含的变量个数. 已有的研究结果已经证明子集选择问题是 NP 难的 [5,18]. 这里只考虑最小化问题, 因为最大化目标函数 f 可以等价地表示成最小化 $-f$.

定义1 (子集选择) 给定所有变量 $V = \{X_1, \ldots, X_n\}$, 一个优化目标 f 以及一个正整数 k, 子集选择问题是去找到一个子集 $S \subseteq V$ 使其满足

$$\arg\min_{S \subseteq V} f(S) \quad \text{s.t.} \quad |S| \leqslant k. \tag{1}$$

稀疏回归问题 [14] 是子集选择问题的一个特例. 如定义 2 所示, 它旨在为线性回归问题找到一个稀疏解. 为了表示方便, 本文不区分一个集合 S 和它包含变量的下标集合 $I_S = \{i \mid X_i \in S\}$. 而且, 假设所有的变量都已被规范化, 即均值为 0, 方差为 1.

算法 1 贪婪算法

算法输入: 所有观测变量 $V=\{X_1,\ldots,X_n\}$, 预测变量 Z 以及正整数 $k\in[1,n]$

算法过程:

1: $t=0$, $S_t=\emptyset$.
2: 重复以下步骤, 直到 $t=k$
3: X^* 是最大化 $R^2_{Z,S_t\cup\{X\}}$ 的一个变量, 也就是说, $X^*=\arg\max_{X\in V\setminus S_t} R^2_{Z,S_t\cup\{X\}}$.
4: $S_{t+1}=S_t\cup\{X^*\}$.
5: $t=t+1$.

算法输出: S_k

定义2 (稀疏回归) 给定所有观测变量 $V=\{X_1,\ldots,X_n\}$, 一个预测变量 Z 和一个正整数 k, 我们定义一个变量子集 $S\subseteq V$ 的均方误差为

$$\mathrm{MSE}_{Z,S}=\min_{\boldsymbol{\alpha}\in\mathbb{R}^{|S|}}\mathrm{E}\left[\left(Z-\sum\nolimits_{i\in S}\alpha_i X_i\right)^2\right].$$

稀疏回归问题是去找一个大小不超过 k 的变量子集使均方误差最小化, 也就是

$$\arg\min_{S\subseteq V}\mathrm{MSE}_{Z,S}\quad\text{s.t.}\quad|S|\leqslant k.$$

对于稀疏回归问题, 之前的理论分析 [13,15,16] 常采用一种等价的表示形式:

$$\arg\max_{S\subseteq V}R^2_{Z,S}\quad\text{s.t.}\quad|S|\leqslant k,\tag{2}$$

其中, $R^2_{Z,S}=(\mathrm{Var}(Z)-\mathrm{MSE}_{Z,S})/\mathrm{Var}(Z)$ 是平方多相关 (squared multiple correlation) 系数 [19,20], 其可以被简化成 $1-\mathrm{MSE}_{Z,S}$, 因为变量 Z 已被规范化, 即方差 $\mathrm{Var}(Z)=1$.

Gilbert 等 [6] 最早研究了一种使用正交匹配跟踪 (OMP) 的二阶段方法在稀疏回归问题上的性能, 证明出在一致性 (coherence) 值 μ 属于 $O(1/k)$ 的条件下, 该方法可以在均方误差 MSE 上取得 $(1+\Theta(\mu k^2))$ - 近似, 其中 μ 是任意两个观测变量之间的最大相关系数. 这个近似性能上界随后被 Tropp 等 [7,21] 进一步改进. 在关于 μ 的相同条件下, Das 和 Kempe [16] 证明出贪婪算法在平方多相关系数 R^2 上可以获得 $(1-\Theta(\mu k))$ - 近似. 如算法 1 所示, 贪婪算法从空集出发, 循环地加入使 R^2 增量最大化的一个变量, 直至选择变量数目达到 k 为止. 然而, 所有这些理论结果都是在 $\mu\in O(1/k)$ 的条件下得到的. 通过引入子模 (submodularity) 比例 γ, Das 和 Kempe [15] 证明了贪婪算法在 R^2 上可以获得 $(1-e^{-\gamma})$ - 近似, 这被认为是目前已知的最佳近似性能, 因为它对 μ 没有任何要求. 除此之外, 一些研究者也尝试分析为达到一定性能所需变量子集大小 k 的下界 [22,23].

2.2 POSS 方法

基于多目标演化算法, Qian 等 [13] 最近提出了一种新型子集选择算法 POSS. 用一个二进制向量 $\boldsymbol{s}\in\{0,1\}^n$ 来表示 V 的一个子集 S, 其中 $s_i=1$ 表示 S 包含变量 X_i, 而 $s_i=0$ 则表示变量 X_i 不出现在 S 中. 为了表示方便, 本文将不区分 $\boldsymbol{s}\in\{0,1\}^n$ 和它对应的子集. POSS 方法的主要思想是将原问题即式 (1) 转化成一个二目标最小化问题求解,

$$\arg\min_{\boldsymbol{s}\in\{0,1\}^n}\ (f_1(\boldsymbol{s}),\ f_2(\boldsymbol{s})),$$

算法 2 POSS 方法

算法输入: 所有变量 $V=\{X_1,\ldots,X_n\}$, 给定目标 f 以及正整数 $k\in[1,n]$

算法参数: 循环轮数 T

算法过程:

1: $\boldsymbol{s}=\{0\}^n$, $P=\{\boldsymbol{s}\}$.
2: $t=0$.
3: 重复以下步骤, 直到 $t=T$
4: 从 P 中均匀随机地选择一个解 $\boldsymbol{s}$.
5: 通过对 $\boldsymbol{s}$ 的每一位以 $1/n$ 的概率翻转 (若当前值为 0, 则变成 1; 反之亦然), 产生一个新解 $\boldsymbol{s}'$.
6: 如果 $\nexists \boldsymbol{z}\in P$ 满足 $\boldsymbol{z}\prec\boldsymbol{s}'$, 那么
7: $Q=\{\boldsymbol{z}\in P\mid \boldsymbol{s}'\preceq\boldsymbol{z}\}$.
8: $P=(P\setminus Q)\cup\{\boldsymbol{s}'\}$.
9: $t=t+1$.

算法输出: $\arg\min_{\boldsymbol{s}\in P,|\boldsymbol{s}|\leqslant k} f_1(\boldsymbol{s})$

其中, 最小化的两个目标 f_1 和 f_2 定义为

$$f_1(\boldsymbol{s})=\begin{cases}+\infty, & \text{如果 } \boldsymbol{s}=\{0\}^n, \text{或者 } |\boldsymbol{s}|\geqslant 2k,\\ f(\boldsymbol{s}), & \text{否则,}\end{cases}\qquad f_2(\boldsymbol{s})=|\boldsymbol{s}|.$$

也就是说, POSS 方法在最小化原始目标函数的同时, 最小化子集大小. 需要注意的是, 子集大小不小于 $2k$ 的解的目标函数值被设置为 $+\infty$, 这是因为这些不可行解违反约束条件 (即 $|\boldsymbol{s}|\leqslant k$) 的程度过大, 对求解原问题几乎没有帮助.

在二目标优化下比较两个解的时候, 它们的两个目标值都需要被比较. POSS 方法采用了一种常用的偏序关系 "支配" (domination). 对于两个解 $\boldsymbol{s}$ 和 $\boldsymbol{s}'$, $\boldsymbol{s}$ 弱支配 $\boldsymbol{s}'$ (也就是说, $\boldsymbol{s}$ 比 $\boldsymbol{s}'$ 好, 记为 $\boldsymbol{s}\preceq\boldsymbol{s}'$) 当且仅当 $f_1(\boldsymbol{s})\leqslant f_1(\boldsymbol{s}')\wedge f_2(\boldsymbol{s})\leqslant f_2(\boldsymbol{s}')$ (也就是说, $\boldsymbol{s}$ 在两个目标函数上的值都不大于 $\boldsymbol{s}'$); $\boldsymbol{s}$ 支配 $\boldsymbol{s}'$ (也就是说, $\boldsymbol{s}$ 比 $\boldsymbol{s}'$ 严格好, 记为 $\boldsymbol{s}\prec\boldsymbol{s}'$) 当且仅当 $\boldsymbol{s}\preceq\boldsymbol{s}'\wedge(f_1(\boldsymbol{s})<f_1(\boldsymbol{s}')\vee f_2(\boldsymbol{s})<f_2(\boldsymbol{s}'))$ (也就是说, $\boldsymbol{s}$ 在一个目标函数上的值不大于 $\boldsymbol{s}'$, 且在另一个目标函数上的值小于 $\boldsymbol{s}'$). 因此, 支配关系可以形式化地定义如下:

(1) $\boldsymbol{s}\preceq\boldsymbol{s}'$, 如果 $f_1(\boldsymbol{s})\leqslant f_1(\boldsymbol{s}')\wedge f_2(\boldsymbol{s})\leqslant f_2(\boldsymbol{s}')$;

(2) $\boldsymbol{s}\prec\boldsymbol{s}'$, 如果 $\boldsymbol{s}\preceq\boldsymbol{s}'\wedge(f_1(\boldsymbol{s})<f_1(\boldsymbol{s}')\vee f_2(\boldsymbol{s})<f_2(\boldsymbol{s}'))$.

如果 $\boldsymbol{s}\preceq\boldsymbol{s}'$ 和 $\boldsymbol{s}'\preceq\boldsymbol{s}$ 均不成立, 则称它们之间不可比. 值得注意的是, Qian 等 [13] 在提出 POSS 方法的时候, 还引入了一个隔离函数 $I:\{0,1\}^n\to\mathbb{R}$: 两个解可以互相比较仅当它们在 I 上取值相同. 但是, 无论是对 POSS 的理论分析还是实验测试, I 都被设置成一个常数函数, 其对于解之间的比较不会有任何影响. 因此, 为了更加清晰地介绍 POSS 方法, 本文将 I 忽略.

接下来, 介绍 POSS 方法的具体过程. 如算法 2 所示, POSS 方法从一个代表空集的解出发 (第 1 行), 循环地改进候选解集 P 中解的质量 (第 3~9 行). 在每一轮循环中, 首先通过对 P 中随机选取的一个解做随机翻转产生一个新解 $\boldsymbol{s}'$ (第 4, 5 行); 然后, $\boldsymbol{s}'$ 被用于更新 P (第 6~8 行): 如果 $\boldsymbol{s}'$ 不被 P 中的任意一个解支配, 它将被加入 P 中, 同时 P 中被 $\boldsymbol{s}'$ 弱支配的解将被删除. 在运行 T 轮后, POSS 方法将终止, 并从 P 中选取满足大小约束的最好解 (即在目标 f_1 上的取值最小) 输出, 作为最终找到的解.

对于稀疏回归问题, Qian 等 [13] 已证明 POSS 方法使用期望运行轮数 $\mathrm{E}[T]\leqslant 2ek^2n$ 能找到一个解 S 满足 $|S|\leqslant k$ 且 $R^2_{Z,S}\geqslant(1-e^{-\gamma})\cdot\mathrm{OPT}$, 这是目前已知的最优近似性能, 之前由贪婪算法获得 [15].

对于稀疏回归问题的一个重要子类 Exponential Decay [16], 他们进一步证明 POSS 方法在运行多项式轮后可以找到一个全局最优解而贪婪算法只能找到局部最优解. 需要注意的是, POSS 方法是随机算法, 故分析得到的结果是基于运行轮数的期望值, 记作 $\mathrm{E}[T]$.

3 DPOSS 方法

本节提出基于分解策略的多目标演化子集选择算法 DPOSS.

从 Qian 等 [13] 对 POSS 方法在稀疏回归问题上的理论分析可见, POSS 方法找到目标解是通过逐步地找到子集大小 $|\boldsymbol{s}|$ 为 0 到 k 的足够好的解来实现的. 将这些中间解记为 $\boldsymbol{s}_0^*, \boldsymbol{s}_1^*, \ldots, \boldsymbol{s}_k^*$, 其中 $|\boldsymbol{s}_i^*| = i$. 在他们的分析中, 解 $\boldsymbol{s}_i^*$ 是在找到 $\boldsymbol{s}_{i-1}^*$ 的基础上, 通过从候选解集 P 中选出 $\boldsymbol{s}_{i-1}^*$ 并翻转其特定的一位来产生的. 由于 POSS 方法在第 4 行中采取均匀随机选择策略, 故从 P 中选出 $\boldsymbol{s}_{i-1}^*$ 的概率为 $1/|P|$. 因为目标 f_1 的定义排除了 $|\boldsymbol{s}| \geqslant 2k$ 的解, 且 P 中任意两个解的 $|\boldsymbol{s}|$ 值都不一样, 所以 $|P|$ 最大可达到 $2k$. 而这正是使得 POSS 方法的计算复杂度 $2ek^2n$ 比贪婪算法的计算复杂度 kn 在渐近上大 k 倍的原因所在. 因此, 一种比较直接的加速策略是: 通过限定 P 的大小, 运行 POSS 方法多次去逐段找到 $\boldsymbol{s}_0^*, \boldsymbol{s}_1^*, \ldots, \boldsymbol{s}_k^*$; 而不再是只运行 POSS 方法一次直接去找到所有解.

基于上述分析, 采取分解策略来逐段找到 $|\boldsymbol{s}| \leqslant k$ 的最优解, 即

$$P^* = \{\boldsymbol{s} \mid \arg\min_{\boldsymbol{s}\in\{0,1\}^n, |\boldsymbol{s}|\leqslant k} (f_1(\boldsymbol{s}),\ f_2(\boldsymbol{s}) = |\boldsymbol{s}|)\}.$$

把 P^* 分解成 m 个子集

$$P_i^* = \{\boldsymbol{s} \mid \arg\min_{\boldsymbol{s}\in\{0,1\}^n, k_{i-1}\leqslant|\boldsymbol{s}|\leqslant k_i} (f_1(\boldsymbol{s}),\ |\boldsymbol{s}|)\}, \quad \forall i = 1, 2, \ldots, m,$$

其中, $k_0 = 0$, $k_m = k$. 为了使 P_i^* 包含的解的个数尽可能接近, 将 k_i 的值设置为

$$\begin{aligned} &\forall 1 \leqslant i \leqslant k - \lfloor k/m \rfloor \cdot m : k_i = k_{i-1} + \lceil k/m \rceil, && \text{从而} \quad |P_i^*| = \lceil k/m \rceil + 1; \\ &\forall k - \lfloor k/m \rfloor \cdot m + 1 \leqslant i \leqslant m : k_i = k_{i-1} + \lfloor k/m \rfloor, && \text{从而} \quad |P_i^*| = \lfloor k/m \rfloor + 1. \end{aligned} \tag{3}$$

在这样的设置下, 容易验证从 $k_0 = 0$ 出发, k_m 最终取值为 k. 而且, 可以发现, 任意相邻的 P_i^* 和 P_{i-1}^* 包含一个共同的大小 $|\boldsymbol{s}| = k_{i-1}$ 的最优解, 且任意 P_i^* 和 P_j^* 包含解的数目最多相差 1.

如算法 3 所示, DPOSS 方法通过调用 POSS 方法依次去找 $P_1^*, P_2^*, \ldots, P_m^*$. 在找 P_i^* 的过程 (称作第 i 阶段) 中, POSS 方法以前一阶段找到的最好解 $\boldsymbol{s}_{k_{i-1}}^*$ 作为初始解, 运行 T_i 轮后, 输出当前找到的最好解 $\boldsymbol{s}_{k_i}^*$. 值得注意的是, 在找 P_i^* 的过程中, 对第一个目标函数 f_1 做相应的修改,

$$f_1(\boldsymbol{s}) = \begin{cases} f(\boldsymbol{s}), & \text{如果 } k_{i-1} \leqslant |\boldsymbol{s}| < 2k_i - k_{i-1}, \\ +\infty, & \text{如果 } |\boldsymbol{s}| \geqslant 2k_i - k_{i-1}. \end{cases}$$

对于 $|\boldsymbol{s}| < k_{i-1}$ 的解, 若在该阶段 POSS 方法的运行中产生, 则将被直接丢弃. 因此, 在找 P_i^* 的过程中, POSS 方法的候选解集 P 包含的解满足 $k_{i-1} \leqslant |\boldsymbol{s}| < 2k_i - k_{i-1}$, 又因为对于任意的 $|\boldsymbol{s}|$ 取值, P 中最多包含一个相应的解, 从而 $|P| \leqslant 2k_i - 2k_{i-1}$, 这比运行 POSS 方法一次直接找 P^* 时的候选解集 P 的大小 $|P| \leqslant 2k$ 要小得多. 另外, 还需要注意的一点是, $\boldsymbol{s}_{k_{i-1}}^*$ 的大小有可能小于 k_{i-1}, 若直接将它作为第 i 阶段的初始解, 它将被丢弃; 对于这种情况, 我们先对解 $\boldsymbol{s}_{k_{i-1}}^*$ 进行扩充: 将任意 $k_{i-1} - |\boldsymbol{s}_{k_{i-1}}^*|$ 个 0 翻转成 1 (即在当前子集中加入任意 $k_{i-1} - |\boldsymbol{s}_{k_{i-1}}^*|$ 个未被选择的变量), 从而使 $|\boldsymbol{s}_{k_{i-1}}^*| = k_{i-1}$, 然后再将它作为第 i 阶段的初始解. 当 $m = 1$ (即不采用分解策略) 时, DPOSS 方法实际上就是 POSS 方法.

算法 3 DPOSS 方法

算法输入: 所有变量 $V=\{X_1,\ldots,X_n\}$, 给定目标 f 以及正整数 $k\in[1,n]$
算法参数: 分解个数 $m\in[1,k]$, 各阶段 POSS 方法运行轮数 $T_1,T_2,\ldots,T_m$
算法过程:

$i=1$.
重复以下步骤, 直到 $i>m$
  如果 $i=1$, 那么
    使用 POSS 方法去找解集 P_1^*, 其中初始解为 $\{0\}^n$, 运行轮数为 T_1.
    在 POSS 方法终止运行后, 输出解为 $\boldsymbol{s}_{k_1}^*=\arg\min_{\boldsymbol{s}\in P,|\boldsymbol{s}|\leqslant k_1} f_1(\boldsymbol{s})$.
  否则,
    使用 POSS 方法去找解集 P_i^*, 其中初始解为 $\boldsymbol{s}_{k_{i-1}}^*$, 运行轮数为 T_i.
    在 POSS 方法终止运行后, 输出解为 $\boldsymbol{s}_{k_i}^*=\arg\min_{\boldsymbol{s}\in P,|\boldsymbol{s}|\leqslant k_i} f_1(\boldsymbol{s})$.
  $i=i+1$.

算法输出: $\boldsymbol{s}_{k_m}^*$

4 理论分析

本节首先从理论上分析了 DPOSS 方法在稀疏回归问题上的性能; 然后通过和 POSS 方法的性能比较, 得出 DPOSS 方法在获得相同近似性能下界的同时, 其所需运行轮数随着分解个数 m 的增加线性下降, 运行时间超线性下降.

4.1 DPOSS 方法性能分析

首先介绍接下来的分析中将要用到的概念子模比例 (submodularity ratio), 其刻画了一个集合函数 f 与子模函数的接近程度, 如定义 3 所示. 我们知道, f 是子模函数当且仅当对于任意 $S_1\subseteq S_2$, $f(S_2)-f(S_1)\leqslant\sum_{X\in S_2\setminus S_1}(f(S_1\cup\{X\})-f(S_1))$ [24], 这等价于对于任意 U 和 k, $\gamma_{U,k}(f)\geqslant 1$ 成立. 当 f 是函数 R^2 时, 简记为 $\gamma_{U,k}$.

定义3 (子模比例 [15]) 令 f 是一个非负集合函数. f 关于集合 U 和参数 $k\geqslant 1$ 的子模比例是

$$\gamma_{U,k}(f)=\min_{L\subseteq U,S:|S|\leqslant k,S\cap L=\emptyset}\frac{\sum_{X\in S}(f(L\cup\{X\})-f(L))}{f(L\cup S)-f(L)}.$$

定理 1 证明了 DPOSS 方法在稀疏回归问题上的近似性能, 其中, $\mathrm{E}[T_i]$ 是 DPOSS 方法各个阶段中调用的 POSS 方法运行轮数的期望值, OPT 是稀疏回归问题即式 (2) 的最优目标函数值. 我们的证明要用到引理 1: 对于任意一个子集, 总是存在一个未被选择的变量, 将其加入该子集带来的 R^2 增量和当前离最优目标函数值 OPT 的距离成比例. 证明思路受 Qian 等 [13] 对 POSS 方法的分析启发而得.

引理1 [13] 对于任意 $S\subseteq V$, 存在一个变量 $\hat{X}\in V-S$ 满足

$$R^2_{Z,S\cup\{\hat{X}\}}-R^2_{Z,S}\geqslant\frac{\gamma_{\emptyset,k}}{k}(\mathrm{OPT}-R^2_{Z,S}).$$

定理1 对于稀疏回归问题, 在参数设置 $\mathrm{E}[T_i]\leqslant 2e(k_i-k_{i-1})^2n$ 下, DPOSS 方法可以找到一个变量子集 S 满足 $|S|\leqslant k$ 且 $R^2_{Z,S}\geqslant(1-e^{-\gamma_{\emptyset,k}})\cdot\mathrm{OPT}$.

证明 首先给出在证明中将被经常用到的一个变量 $J_{\max}$ 的定义. 令 $J_{\max}$ 表示满足如下条件的 $j\in[0,k]$ 的最大值: 在候选解集 P 中, 存在一个解 $\boldsymbol{s}$ 满足 $|\boldsymbol{s}|\leqslant j$ 且 $R^2_{Z,\boldsymbol{s}}\geqslant(1-(1-\frac{\gamma_{\emptyset,k}}{k})^j)\cdot\mathrm{OPT}$. 也

就是说,

$$J_{\max}=\max\left\{j\in[0,k]\mid\exists\boldsymbol{s}\in P,|\boldsymbol{s}|\leqslant j\wedge R^2_{Z,\boldsymbol{s}}\geqslant\left(1-\left(1-\frac{\gamma_{\emptyset,k}}{k}\right)^j\right)\cdot\mathrm{OPT}\right\}.$$

DPOSS 方法的第 i 阶段 (即找 P^*_i 的过程) 在第 $i-1$ 阶段结束后开始. 我们规定: 在第 i 阶段中, 当 $J_{\max}=k_i$ 时, 该阶段结束. 因此, 当第 m 阶段结束时, $J_{\max}=k_m=k$, 这意味着我们已经找到了期望的一个解, 因为此时 P 中存在一个解 $\boldsymbol{s}$ 满足 $|\boldsymbol{s}|\leqslant k$ 且 $R^2_{Z,\boldsymbol{s}}\geqslant(1-(1-\frac{\gamma_{\emptyset,k}}{k})^k)\cdot\mathrm{OPT}\geqslant(1-e^{-\gamma_{\emptyset,k}})\cdot\mathrm{OPT}$, 其中第 2 个不等号由 $(1-\frac{\gamma_{\emptyset,k}}{k})^{\frac{k}{\gamma_{\emptyset,k}}}\leqslant\frac{1}{e}$ 可推得. 因此, 只需要分析各个阶段从开始到结束的运行轮数即可.

接下来分析在第 i 阶段中使得 $J_{\max}=k_i$ 的期望运行轮数 $\mathrm{E}[T_i]$. 这一阶段的初始解是第 $i-1$ 阶段结束时找到的最优解 $\boldsymbol{s}^*_{k_{i-1}}$, 由我们对每一阶段结束要满足的条件易得

$$|\boldsymbol{s}^*_{k_{i-1}}|\leqslant k_{i-1},\qquad R^2_{Z,\boldsymbol{s}^*_{k_{i-1}}}\geqslant\left(1-\left(1-\frac{\gamma_{\emptyset,k}}{k}\right)^{k_{i-1}}\right)\cdot\mathrm{OPT}.$$

又因为 $R^2_{Z,\boldsymbol{s}}$ 单调递增, 所以将解 $\boldsymbol{s}^*_{k_{i-1}}$ 扩充至包含 k_{i-1} 个变量的一个解将不影响上述公式的成立. 因此, 在此阶段中, $J_{\max}$ 的初始值至少为 k_{i-1}. 假定当前 $J_{\max}=i<k_i$. 令 $\boldsymbol{s}$ 是和取值 i 对应的一个解, 也就是说, $|\boldsymbol{s}|\leqslant i$ 且 $R^2_{Z,\boldsymbol{s}}\geqslant(1-(1-\frac{\gamma_{\emptyset,k}}{k})^i)\cdot\mathrm{OPT}$. 易见, $J_{\max}$ 不可能减小. 若 $\boldsymbol{s}$ 保留在 P 中, 这显然成立. 若从 P 中把 $\boldsymbol{s}$ 删除 (见算法 2 的第 7, 8 行), 那么弱支配 $\boldsymbol{s}$ 的新产生解 $\boldsymbol{s}'$ (即 $\boldsymbol{s}'$ 拥有更少的变量和更大的 R^2 值) 将被加入 P 中, 这使得 $J_{\max}\geqslant i$, 故 $J_{\max}$ 不减小. 由引理 1 可知, 通过翻转 $\boldsymbol{s}$ 的一个特定 0 位 (也就是在相应的子集 S 中加入一个特定变量) 能够产生一个新解 $\boldsymbol{s}'$ 满足 $R^2_{Z,\boldsymbol{s}'}-R^2_{Z,\boldsymbol{s}}\geqslant\frac{\gamma_{\emptyset,k}}{k}(\mathrm{OPT}-R^2_{Z,\boldsymbol{s}})$. 故,

$$R^2_{Z,\boldsymbol{s}'}\geqslant\left(1-\frac{\gamma_{\emptyset,k}}{k}\right)R^2_{Z,\boldsymbol{s}}+\frac{\gamma_{\emptyset,k}}{k}\cdot\mathrm{OPT}\geqslant\left(1-\left(1-\frac{\gamma_{\emptyset,k}}{k}\right)^{i+1}\right)\cdot\mathrm{OPT},$$

其中第 2 个不等号由 $R^2_{Z,\boldsymbol{s}}\geqslant(1-(1-\frac{\gamma_{\emptyset,k}}{k})^i)\cdot\mathrm{OPT}$ 可推得. 因为 $|\boldsymbol{s}'|=|\boldsymbol{s}|+1\leqslant i+1$, $\boldsymbol{s}'$ 必将被加入 P 中; 否则, 由算法 2 第 6 行可见, $\boldsymbol{s}'$ 被 P 中的一个解支配, 这意味着此时 $J_{\max}$ 已经大于 i, 和假设 $J_{\max}=i$ 相矛盾. 在加入 $\boldsymbol{s}'$ 后, $J_{\max}\geqslant i+1$. 令 $P_{\max}$ 表示该阶段中 POSS 方法的候选解集 P 包含解的个数 (即 $|P|$) 的最大值. 由以上分析可得, $J_{\max}$ 在一轮运行中增大至少 1 的概率至少为

$$\frac{1}{P_{\max}}\cdot\frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1}\geqslant\frac{1}{enP_{\max}},$$

其中 $\frac{1}{P_{\max}}$ 是算法 2 的第 4 行中选择 $\boldsymbol{s}$ 的概率的下界, $\frac{1}{n}(1-\frac{1}{n})^{n-1}$ 是在第 5 行中翻转 $\boldsymbol{s}$ 的一个特定位而保持其他位不变的概率. 那么, $J_{\max}$ 增大至少 1 所需期望轮数最多是 $enP_{\max}$. 因此, 在 $(k_i-k_{i-1})\cdot enP_{\max}$ 期望轮数后, $J_{\max}$ 取值达到 k_i, 也就是说, $\mathrm{E}[T_i]\leqslant(k_i-k_{i-1})\cdot enP_{\max}$.

然后分析 $P_{\max}$ 的上界. 由 POSS 方法的过程可知, 候选解集 P 包含的解是不可比的. 因此, 一个目标的每个可能取值在 P 中存在最多一个对应解. 因为满足 $|\boldsymbol{s}|\geqslant 2k_i-k_{i-1}$ 的解在第 1 个目标上的取值为 $+\infty$, 所以这些解将被排除出 P. 又因为满足 $|\boldsymbol{s}|<k_{i-1}$ 的解将被直接丢弃, 故第 2 个目标的可能取值为 $|\boldsymbol{s}|\in\{k_{i-1},k_{i-1}+1,\ldots,2k_i-k_{i-1}-1\}$, 这意味着 $P_{\max}\leqslant 2k_i-2k_{i-1}$. 因此, 第 i 阶段的期望运行轮数 $\mathrm{E}[T_i]\leqslant 2e(k_i-k_{i-1})^2n$. 故, 该定理成立.

4.2 比较 DPOSS 方法和 POSS 方法

将 DPOSS 方法总的运行轮数记为 T, 即 $T=T_1+T_2+\cdots+T_m$. 由定理 1 可知, DPOSS 方法获

得 $(1-e^{-\gamma_{\emptyset,k}})$ - 近似性能所需期望轮数为

$$\mathrm{E}[T]=\mathrm{E}[T_1+\cdots+T_m]=\sum_{i=1}^{m}\mathrm{E}[T_i]\leqslant\sum_{i=1}^{m}2e(k_i-k_{i-1})^2n\leqslant 2en\left\lceil\frac{k}{m}\right\rceil\sum_{i=1}^{m}(k_i-k_{i-1})=2ekn\left\lceil\frac{k}{m}\right\rceil,$$

其中, 第 2 个等号由期望的线性可加性可得, 第 2 个不等号由对 k_i 的设置即式 (3) 易得, 最后一个等号是因为 $k_0=0$, $k_m=k$.

Qian 等 [13] 证明 POSS 方法在期望轮数 $\mathrm{E}[T]\leqslant 2ek^2n$ 下获得 $(1-e^{-\gamma_{\emptyset,k}})$ - 近似性能. 通过和 POSS 方法性能的比较, 我们发现采用分解策略的 DPOSS 方法在获得相同近似性能下界的同时, 所需运行轮数随着分解个数 m 的增加几乎线性下降, 即 $2ek^2n/(2ekn\lceil\frac{k}{m}\rceil)\approx m$.

接下来, 进一步比较 POSS 方法和 DPOSS 方法每一轮的运行时间 t_{p} 和 t_{dp}. 它们每一轮的运行时间主要是耗费在对新产生解 $\boldsymbol{s}'$ 的目标函数评估上, 其代价通常取决于 $\boldsymbol{s}'$ 包含 1 的数目 (即相应子集包含变量的数目). 为了分析方便, 不妨假设线性依赖关系, 即对 $\boldsymbol{s}'$ 的目标函数评估时间为 $c\cdot|\boldsymbol{s}'|$, 其中 c 为一个常数. 由上述的分析可知, 对于子集大小 $|\boldsymbol{s}|\in\{0,1,\ldots,2k-1\}$ 的每个可能取值, POSS 方法的候选解集 P 最多包含一个解, 而对于 $|\boldsymbol{s}|\geqslant 2k$, P 不包含任何解. 为了可分析性, 假设 P 对于任意 $|\boldsymbol{s}|\in\{0,1,\ldots,2k-1\}$ 都包含了一个相应的解. 由于 POSS 方法第 4 行均匀随机地选取一个解 $\boldsymbol{s}$, 故 $|\boldsymbol{s}|=j\in\{0,1,\ldots,2k-1\}$ 的概率是 $1/2k$. 又因为解 $\boldsymbol{s}'$ 通过对 $\boldsymbol{s}$ 的每一位以 $1/n$ 的概率翻转产生 (第 5 行), 易得 $\mathrm{E}[|\boldsymbol{s}'|]=|\boldsymbol{s}|+1-2|\boldsymbol{s}|/n$. 因此, 可以分析得出

$$\mathrm{E}[t_{\mathrm{p}}]=\sum_{j=0}^{2k-1}\frac{1}{2k}\cdot\mathrm{E}[t_{\mathrm{p}}\mid|\boldsymbol{s}|=j]=\sum_{j=0}^{2k-1}\frac{1}{2k}\cdot c\cdot\left(j+1-\frac{2j}{n}\right)=c\cdot\left(\left(1-\frac{2}{n}\right)k+\frac{1}{2}+\frac{1}{n}\right).$$

对于 DPOSS 方法, 首先分析它在每个阶段中每一轮的运行时间, 记为 t_{dp}^i $(1\leqslant i\leqslant m)$. 在第 i 阶段中, 根据定理 1 的证明过程可知: 对于子集大小 $|\boldsymbol{s}|\in\{k_{i-1},k_{i-1}+1,\ldots,2k_i-k_{i-1}-1\}$ 的每个可能取值, 所调用 POSS 方法的候选解集 P 最多包含一个解, 而对于任意其他的 $|\boldsymbol{s}|$, P 不包含任何解. 和对 t_{p} 的分析相同, 我们假设 P 对于任意 $|\boldsymbol{s}|\in\{k_{i-1},k_{i-1}+1,\ldots,2k_i-k_{i-1}-1\}$ 都包含了一个相应的解. 故, 同样可以分析得出

$$\begin{aligned}\mathrm{E}[t_{\mathrm{dp}}^i]&=\sum_{j=k_{i-1}}^{2k_i-k_{i-1}-1}\frac{1}{2(k_i-k_{i-1})}\cdot\mathrm{E}[t_{\mathrm{dp}}^i\mid|\boldsymbol{s}|=j]\\&=\sum_{j=k_{i-1}}^{2k_i-k_{i-1}-1}\frac{1}{2(k_i-k_{i-1})}\cdot c\cdot\left(j+1-\frac{2j}{n}\right)=c\cdot\left(\left(1-\frac{2}{n}\right)k_i+\frac{1}{2}+\frac{1}{n}\right).\end{aligned}$$

为了分析方便, 不妨假设 k 是 m 的整数倍, 故 $\forall 1\leqslant i\leqslant m: k_i=i\cdot\frac{k}{m}$. 通过将所有阶段综合考虑, 可以分析得出

$$\mathrm{E}[t_{\mathrm{dp}}]=\sum_{i=1}^{m}\frac{1}{m}\cdot\mathrm{E}[t_{\mathrm{dp}}^i]=\sum_{i=1}^{m}\frac{1}{m}\cdot c\cdot\left(\left(1-\frac{2}{n}\right)k_i+\frac{1}{2}+\frac{1}{n}\right)=c\cdot\left(\left(1-\frac{2}{n}\right)\left(\frac{1}{2}+\frac{1}{2m}\right)k+\frac{1}{2}+\frac{1}{n}\right),$$

其中第 1 个等号是因为每个阶段的运行轮数 $2e(k_i-k_{i-1})^2n=2ek^2n/m^2$ 相等.

因此, 在一定的假设下, 分别分析得出了 POSS 方法和 DPOSS 方法每一轮的期望运行时间 $\mathrm{E}[t_{\mathrm{p}}]$ 和 $\mathrm{E}[t_{\mathrm{dp}}]$. 通过比较发现, $\mathrm{E}[t_{\mathrm{dp}}]$ 要小于 $\mathrm{E}[t_{\mathrm{p}}]$. 结合 DPOSS 方法在运行轮数上获得关于分解个数 m 的线性加速比这一点, 得出 DPOSS 方法在运行时间上可以获得关于 m 的超线性加速比.

表 1 实验用数据集信息

Table 1 The data sets

Data set	Instances	Features	Data set	Instances	Features	Data set	Instances	Features
Eunite2001	367	16	Sonar	208	60	Clean1	476	166
Svmguide3	1284	21	Triazines	186	60	Gisette	7000	5000
Ionosphere	351	34	Coil2000	9000	86			

5 实验测试

本节通过在稀疏回归问题上的实验测试了 DPOSS 方法的实际性能. 首先介绍实验的设置, 然后给出实验的结果.

5.1 实验设置

在 8 个数据集 [1)]上测试了 DPOSS 方法的性能, 关于数据集的具体信息由表 1 给出. 对于稀疏回归问题的稀疏度, 将 k 设置为 8; 对于 DPOSS 方法的参数 $T_1, T_2, \ldots, T_m$, 使用定理 1 中分析得出的取值. 我们将评估 $m = 1, 2, 3, 4$ 时 DPOSS 方法获得的运行时间加速比以及输出解的 R^2 值. 当 $m = i$ 时, 加速比的计算如下:

$$\text{加速比} = \frac{\text{POSS 方法的 CPU 时间}}{\text{DPOSS 方法 } (m = i) \text{ 的 CPU 时间}}.$$

需要注意的是, DPOSS 方法在 $m = 1$ 时就是 POSS 方法. 由于 DPOSS 方法是随机算法, 对于 m 的每个取值和每个数据集, 独立地运行 DPOSS 方法 10 次, 报告平均加速比和 R^2 的平均值. 同时也和贪婪算法找到解的 R^2 值进行了比较. 贪婪算法是在 POSS 方法之前在稀疏回归问题上获得目前已知最佳近似性能的算法 [15].

本文所有方法用 Matlab 2013 实现, 所有实验在同一台服务器上完成. 服务器配置如下: 内存 32 GB, 4 个 Intel Xeon CPU, 每个 CPU 有 8 个核, 每个核主屏 2.00 GHz. 服务器操作系统是 Red Hat 4.1.2-48, 内核版本是 Linux 2.6.18-194.el5.

5.2 实验结果

图 1 画出了全部实验结果. 从每个数据集上的左子图可以直观地观察到, DPOSS 方法在运行时间上的加速比曲线总是在线性加速比曲线 (即实线) 上方, 这和我们的理论结果相一致. 通过每个数据集上的右子图, 可以看到, DPOSS 方法输出解的 R^2 值随着分解个数 m 的增加会有所下降, 但始终比贪婪算法输出解的 R^2 值要好. 需要注意的是, 在某些数据集 (如 svmguide3) 和特定的 m 值 (如 $m = 1$) 上, DPOSS 方法输出解的 R^2 值的标准差为 0, 这是因为 DPOSS 方法的 10 次独立运行均找到了相同质量的解. 还需要强调的一点是, DPOSS 方法输出解的 R^2 值随 m 增加有所下降与我们的理论结果并不矛盾, 因为我们理论上仅证明了 DPOSS 方法获得的近似性能下界关于 m 保持不变, 其实际性能有可能变差. 这也意味着目前得出的 DPOSS 方法的近似性能下界可能仍有进一步改进的空间.

1) 这些数据集来源于 http://archive.ics.uci.edu/ml/ 和 http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

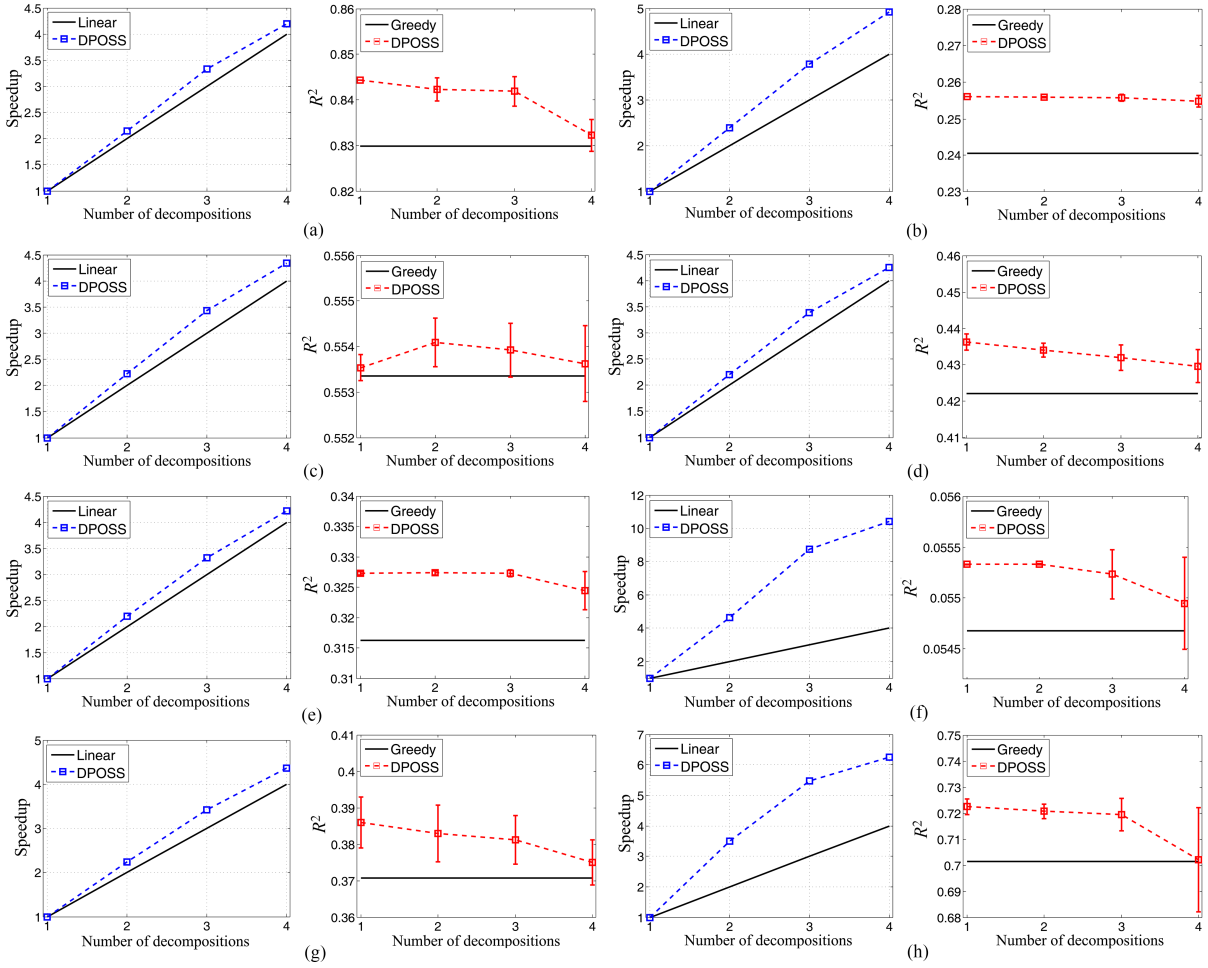

图 1 (网络版彩图) 在每一个数据集上, 左子图: 运行时间加速比的平均值, 右子图: 输出解的 R^2 值的平均值和标准偏差 (R^2 值越大越好)

Figure 1 (Color online) On each data set, left: speedup, right: the average and standard deviation of the R^2 value (the larger the better). (a) Eunite2001; (b) Svmguide3; (c) Ionosphere; (d) Sonar; (e) Triazines; (f) Coil2000; (g) Clean1; (h) Gisette

6 结束语

子集选择问题是在许多领域中经常遇到的一类 NP 难问题. 最近提出的一种基于多目标演化算法的子集选择算法 POSS 在理论和实验上都获得了目前的最佳性能, 然而, 其运行时间在问题规模很大时变得无法令人满意. 在 POSS 方法的基础上, 通过将整个子集空间分解成多个子空间逐步求解, 本文提出了一种基于分解策略的多目标演化子集选择算法 DPOSS. 在理论上, 证明了 DPOSS 方法在获得和 POSS 方法相同近似性能下界的同时, 运行时间随着分解个数的增加超线性下降. 实验结果验证了我们关于运行时间超线性下降的理论结果, 并显示出 DPOSS 方法的实际性能随着分解个数的增加略有下降, 这意味着目前对于 DPOSS 方法近似性能下界的分析可能仍有进一步改进的空间.

参考文献

1 Gu M, Eisenstat S C. Efficient algorithms for computing a strong rank-revealing QR factorization. SIAM J Sci Comput, 1996, 17: 848–869

2 Zhou Z-H. Ensemble Methods: Foundations and Algorithms. Boca Raton: Chapman and Hall/CRC, 2012

3 Xie B, Liu Y, Zhang H, et al. Efficient image representation for object recognition via pivots selection. Front Comput Sci, 2015, 9: 383–391

4 Zhang Y Q, Xiao J S, Li S H, et al. Learning block-structured incoherent dictionaries for sparse representation. Sci China Inf Sci, 2015, 58: 102302

5 Davis G, Mallat S, Avellaneda M. Adaptive greedy approximations. Constr Approx, 1997, 13: 57–98

6 Gilbert A C, Muthukrishnan S, Strauss M J. Approximation of functions over redundant dictionaries using coherence. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, 2003. 243–252

7 Tropp J A. Greed is good: algorithmic results for sparse approximation. IEEE Trans Inf Theory, 2004, 50: 2231–2242

8 Tibshirani R. Regression shrinkage and selection via the lasso. J Royal Stat Soc: Ser B (Methodological), 1996, 58: 267–288

9 Zou H, Hastie T. Regularization and variable selection via the elastic net. J Royal Stat Soc: Ser B (Methodological), 2005, 67: 301–320

10 Yu Y, Yao X, Zhou Z-H. On the approximation ability of evolutionary optimization with application to minimum set cover. Artif Intell, 2012, 180–181: 20–33

11 Qian C, Yu Y, Zhou Z-H. Pareto ensemble pruning. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence, Austin, 2015. 2935–2941

12 Qian C, Yu Y, Zhou Z-H. On constrained Boolean Pareto optimization. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence, Buenos Aires, 2015. 389–395

13 Qian C, Yu Y, Zhou Z-H. Subset selection by Pareto optimization. In: Proceedings of Advances in Neural Information Processing Systems 28, Montreal, 2015. 1765–1773

14 Miller A. Subset Selection in Regression. 2nd ed. London: Chapman and Hall/CRC, 2002

15 Das A, Kempe D. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In: Proceedings of the 28th International Conference on Machine Learning, Bellevue, 2011. 1057–1064

16 Das A, Kempe D. Algorithms for subset selection in linear regression. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, 2008. 45–54

17 Qian C, Shi J-C, Yu Y, et al. Parallel Pareto optimization for subset selection. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence, New York, 2016. 1–7

18 Natarajan B K. Sparse approximate solutions to linear systems. SIAM J Sci Comput, 1995, 24: 227–234

19 Diekhoff G. Statistics for the Social and Behavioral Sciences: Univariate, Bivariate, Multivariate. New York: William C Brown Pub, 1992

20 Johnson R A, Wichern D W. Applied Multivariate Statistical Analysis. 6th ed. New York: Pearson, 2007

21 Tropp J A, Gilbert A C, Muthukrishnan S, et al. Improved sparse approximation over quasiincoherent dictionaries. In: Proceedings of the 11th International Conference on Image Processing, Barcelona, 2003. 37–40

22 Shalev-Shwartz S, Srebro N, Zhang T. Trading accuracy for sparsity in optimization problems with sparsity constraints. SIAM J Optimiz, 2010, 20: 2807–2832

23 Yuan X-T, Yan S. Forward basis selection for pursuing sparse representations over a dictionary. IEEE Trans Pattern Anal Mach Intell, 2013, 35: 3025–3036

24 Nemhauser G L, Wolsey L A, Fisher M L. An analysis of approximations for maximizing submodular set functions - I. Math Prog, 1978, 14: 265–294

Decomposition-based Pareto optimization for subset selection

Chao QIAN[1,2,3] & Zhi-Hua ZHOU[1,2*]

1 *National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210023*, China*;
2 *Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing* 210023*, China*;
3 *School of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027*, China*
*E-mail: zhouzh@nju.edu.cn

Abstract In many machine-learning tasks, subset selection, which selects a few variables from a large set, is a fundamental problem; it is, however, NP-hard. The recently emerged Pareto Optimization for Subset Selection (POSS) method is a powerful approximation solver for this problem. However, the POSS running time can be unsatisfactory when the problem size is large, restricting its large-scale applications. In this paper, we propose the DPOSS method, which uses a decomposition strategy. DPOSS decomposes the entire subset space into several subspaces, and then sequentially applies the POSS method. Our theoretical analysis shows that DPOSS can achieve the same approximation guarantee as POSS, while superlinearly reducing its running time with respect to the number of decompositions. Empirical studies show that DPOSS's actual running time decreases superlinearly, and the quality of the produced solution has a little loss. However, it is still better than the greedy algorithm, the previous algorithm with the best known theoretical guarantee.

Keywords machine learning, subset selection, multi-objective optimization, multi-objective evolutionary algorithm, decomposition

Chao QIAN was born in 1987. He received his Ph.D. degree in Computer Science from Nanjing University in 2015. Currently, he is an associate researcher at the University of Science and Technology of China. His research interests include artificial intelligence, machine learning, and data mining.

Zhi-Hua ZHOU was born in 1973. He received his Ph.D. degree in Computer Science from Nanjing University, China, in 2000. Currently, he is a professor at Nanjing University. His research interests mainly include artificial intelligence, machine learning, and data mining. He is a fellow of the AAAI, IEEE, IAPR, IET/IEE, and CCF, and is an ACM Distinguished Scientist.