Sparse Canonical Correlation Analysis B&B Algorithm

Quill Healey

May 6, 2024

Introduction and Notation

Let $\mathcal{P}_{\text{init}}$ denote the subset selection maximization problem we wish to solve. More specifically, we have two vectors of boolean optimization variables of lengths n_1 and n_2. We must select $s_1 < n_1$ and $s_2 < n_2$ of these variables, respectively, to be "true." Or equivalently, if the variables are $x \in \{0,1\}^{n_1}$ and $y \in \{0,1\}^{n_2}$, the problem is to find $x_i = 1$ for $i = 1, \ldots, s_1$ and $y_i = 1$ for $i = 1, \ldots s_2$ to *maximize* the objective function. We'll let $p^* = \sup\{\text{objective of SCCA problem}\}$. To globally solve this problem, we utilize the following branch and bound algorithm.

(Please note that I use the branch and bound notation from [1] *and* I blatantly steal their explanation logic/ordering, just modified to match this problem.)

Subproblem Fixed Variables Notation

As with all branch and bound methods, our algorithm works by "fixing" variables "into" and "out of" the tree. The specifics of this fixing will be specified below. Here, we simply wish to emphasize that because we have constraints on the number of variables we wish to select, while executing the algorithm it is important to keep track of $s_i' \leq s_i$ and $l_i' \leq n_i - s_i$, $i = 1, 2$, which we'll define as the number of variables fixed into or out of, respectively, a given subproblem. Because it is unecessary for the sake of understanding the branch and bound algorithm to develop more specific notation, we will not do so. Consequently, please keep in mind that when we refer to some subproblem's s_i' or l_i', $i = 1, 2$, the subproblem's actual fixed variables will differ from *any other subproblem's* fixed variables.

Solving

The branch and bound algorithm we describe here finds p^*. Before describing the algorithm, we establish some more notation. Firstly, for any problem, $\mathcal{P}$, in our branch and bound tree we define

$$\Phi_{\max}(\mathcal{P}) = \sup_{x,y}\{\mathcal{P} \text{ objective}\},$$

so $p^* = \Phi_{\max}(\mathcal{P}_{\text{init}})$.

The algorithm will **primarily** rely on the function $\Phi_{\text{ub}}(\mathcal{P})$, defined for any problem of $\mathcal{P}$, which provides the *relaxed objective value* associated with forcing that problem's optimization variables to be true or false according to the respective, associated lists of sizes s'_i and l'_i. Importantly, note that $\Phi_{\text{ub}}(\mathcal{P})$ **does not provide** the associated relaxed solution. We also have access to $\Phi_{\text{lb}}(\mathcal{P})$, which provides a *feasible lower bound solution* to any $\mathcal{P}$. However, because this lower bound algorithm is expensive to compute, it's only practical to use for the initial problem. It follows that

$$\Phi_{\text{lb}}(\mathcal{P}) \leq \Phi_{\max}(\mathcal{P}) \leq \Phi_{\text{ub}}(\mathcal{P})$$

for any problem $\mathcal{P}$.

The algorithm is as follows. We start by computing

$$L_1 = \Phi_{\text{lb}}(\mathcal{P}_{\text{init}}), \quad U_1 = \Phi_{\text{ub}}(\mathcal{P}_{\text{init}}),$$

which are lower and upper bounds on p^*, respectively. If $U_1 - L_1 \leq \varepsilon$, the algorithm terminates. Otherwise, randomly select an index $k = 1, 2, \ldots n_1, n_1 + 1, \ldots, n_2 - 1, n_2$, and form two problems: (WLOG assume $k \leq n_1$) The first problem will require that $x_k = 0$ while the second problem will require that $x_k = 1$. Consequently, subproblem one will have $s'_1 = 1$ and subproblem two will have $l'_1 = 1$. Each of these problems is the same subset maximizing selection problem as before, but each now with only $n_1 + n_2 - 1$ boolean variables.

We now solve the two relaxations of these subproblems to obtain a new upper bound on the optimal value of each subproblem. We'll denote these as $\tilde{U}$ (for $x_k = 0$) and $\bar{U}$ (for $x_k = 1$). Each of these two upper bounds must be at least as small as U_1, i.e., $\max\{\tilde{U}, \bar{U}\} \leq U_1$. Furthermore, we obtain the following bounds on p^*:

$$L_1 = \Phi_{\text{lb}}(\mathcal{P}) \leq p^* = U_2 = \max\{\tilde{U}, \bar{U}\}.$$

By the inequalities above, we have $U_2 - L_1 \leq U_1 - L_1$.

At the next step we can choose to split either of the subproblems. After selecting one, we need to split that problem on an index not equal to k, the index we used to split the original problem. We also must check if fixing a variable into or out of the problem will violate a cardinality constraint. Given these constraints are not violated, we can perform this random splitting, solve the relaxations for the split subproblem (which have $n_1 + n_2 - 2$ boolean variables), and obtain new upper bounds for each.

At this point we have formed a partial binary tree of subproblems. The root is the original problem; the first split yields two child subproblems, one with $x_k = 0$ and the other with $x_k = 1$. The second iteration yields another two children of one of the original children. We continue this way. At each iteration, we choose a leaf node (which corresponds to a subproblem with some of the boolean variables fixed to particular values), and so long as no cardinality constraints will be violated, split it by fixing a varaible that is not fixed in the parent problem.

The maximum of the upper bounds, over all leaf nodes, gives a upper bound on the optimal value p^*. We refer to this upper bound as U_B. Importantly, because we do not

have a cheap lower bound method and Φ_{ub} does not provide a relaxed solution (which if it did, a relaxed solution could happen to contain all boolean values), the lower bound is only updated if we search an entire tree branch and that terminal leaf node's feasible solution has an objective value that is greater than L_B (which remains L_1 until such a terminal node is found).

Despite our inability to update the lower bound, we can carry out pruning (to save memory, but not time) as the algorithm progresses. Specifically, if a subproblem's upper bound is less than L_B, this subproblem can be removed from the tree since all points corresponding to that node are worse than the current best point found.

Subproblem picking I'll implement two tactics: DFS and choosing the subproblem to shrink the UB.

Variable Fixing: Use scores.

The Formal Algorithm

For the sake of clarity, the following algorithm does not explicitly include the logic required to ensure that the cardinality constraints $s_1' \leq s_1$ and $s_2' \leq s_2$ are respected. This logic is instead implicit in the statement "split $\mathcal{P}$ validly using S_1 and S_0," where S_1 and S_0 contain integer values $i = 0, 1, \ldots, n_1{-}1, n_1, \ldots, n_2{-}2, n_2{-}1$ (assuming the algorithm is implemented in a zero-indexed programming language) corresponding to fixing a corresponding variable to one or zero, respectively. For more on how this logic works, see the variable fixing section below.

Algorithm 1. Branch and Bound Algorithm

$k = 0$;
$\mathcal{L}_k = \{\mathcal{P}_{\text{init}}\}$;
$L_B = \Phi_{\text{lb}}(\mathcal{P}_{\text{init}})$;
$U_B = \Phi_{\text{ub}}(\mathcal{P}_{\text{init}})$;
$S_0, S_1 = \text{varfix}(\mathcal{P}_{\text{init}})$;
while $U_B - L_B > \varepsilon$ **do**
  Pick $\mathcal{P}_k \in \mathcal{L}_k$;
  Split $\mathcal{P}$ validly, using S_1 and S_0, into $\mathcal{P}_I$ and $\mathcal{P}_{II}$;
  Form $\mathcal{L}_{k+1}$ from $\mathcal{L}_k$ by removing $\mathcal{P}_k$ and adding $\mathcal{P}_I$ and $\mathcal{P}_{II}$;
  $L_B = \max\{\Phi_{\text{lb}}(\mathcal{P}_{\text{init}}), \text{values of exhausted branches}\}$;
  $U_B = \max_{\mathcal{P} \in \mathcal{L}_{k+1}} \Phi_{\text{ub}}(\mathcal{P})$;
  $k = k + 1$;
end
Optimal solution is a terminal leaf node or a certificate verifying that $\Phi_{\text{lb}}(\mathcal{P}_{\text{init}})$ is optimal;

Variable Fixing

For a given subproblem, recall that s_1' is the number of chosen elements of x and s_2' is the number of chosen elements of y (chosen $\Leftrightarrow$ elements known to be one). Conversely, l_1' is the

number of discarded elements of x and l_2' is the number of discarded elements of y (discarded $\Leftrightarrow$ elements known to be zero). A terminal leaf node (a leaf node of the fully enumerated SCCA tree) can then be characterized in two ways. The obvious one is that the desired s_i' elements have been chosen $\Leftrightarrow s_i' = s_i$, $i = 1, 2$. The slightly more nuanced characterization is that $l_i' = n_i' - s_i$ where $n_i' = n_i - s_i'$, $i = 1, 2$. When implementing the branch and bound tree, it is tempting to conclude that the equality $l_i' = n_i - s_i$ is a terminal leaf node certificate; however, this equality fails to account for the elements which are known to be chosen. Since we can only discard elements which are not already chosen, the number of possible elements to discard must be equal to the original number of elements which could be discarded $(n_i - s_i)$ *less* the number of elements which are chosen $(n_i - s_i - s_i')$.

References

[1] Stephen Boyd and Jacob Mattingley. Lecture notes on branch and bound methods, ee364b, 2018.