covtree Pseudocode / functional overview

Caution: This code is not written to commercial standards, it is designed for a high-performance but limited scope use case in the FitCons2 model.

covtree is an optimized implementation of decision tree analysis from the CART family of methodologies. The algorithm begins with a specified set of autosomal genomic positions (by default the 2.88 billion autosomal positions in hg19), and identifies a single covariate bi-partitioning of positions (or weighted fraction of a position) that maximizes post-split likelihood, with likelihood for each child calculated under a separately fitted Insight2 model. After a split, this process is revisited recursively for positions in each child until the best split fails to meet a minimum likelihood improvement.

A custom implementation was required to accommodate the large data set (9 covariates for each of 2.9 billion positions in each of 115 cell types), and computationally demanding inference function. For efficiency, covariates are quantized and bit fields are used to represent values of each quantized covariate. This also employs instruction level parallelism at the bit level, allowing each 64-bit instruction to make 64, 1-bit comparisons in each machine cycle. Even at this level of optimization, loading the complete data set requires approximately 6GB of data and can take several minutes, while traversing the database and writing partitions for all covariates can require several hours.

Functional Overview:

Process user arguments

> including recursion mode flag: 0-recurse internally, 2-write results and exit, allow external logic to submit child jobs in parallel, 1-one child is externally processed, the other internally

Initialize runtime elements (runtimeElements::init())

- Initialize Insight2 processes in server mode, identify input and output directories
- Read the database definition file and convert to bitfield template
- Load the covariate database – if a cached copy, read the array of start, length, class values as a single binary read, otherwise parse ".bedg" file and write .cache file for later use. (covdb.loadDB())
- Identify the root / parent set of genomic positions U – by default hg19 autosome. Initialize linear mapping that assigns each genomic position (chrom Pos) a unique, sequential 32 bit unsigned integer

Reorder nonmonotonic covariate values based on insight ρ values (covDefST())

- For each non-monotonic covariate (C , with $|C| = N$ having values $C_i \in \{1,2,3 \dots N\}$), partition universal (parent) set according to values of the nonmonotonic covariate.
- Generate Insight2 score (ρ_i) for positions associated with each covariate value i (saved to ./sing/*)
- Reorder covariate values to corresponding to increasing values of ρ_i .
- Save summary results to Save sorted partition information in ./covtree.outsing.txt

For each ordered covariate C with $|C| = N$ having values $C_i \in \{1,2,3 \dots N\}$, (main::evaluatePartitions())

- identify the set of all ordered bi-patriations on that covariate with non-null parts.(scorePair::GenerateCosets())
  - For example, for $N = 5$ the set of 4 partitions $\{P_1\}\{P_2\}$ [for co-sets P_1, P_2] are
  - $\{C_1\}\{C_2, C_3, C_4, C_5\}$, $\{C_1, C_2\}\{C_3, C_4, C_5\}$, $\{C_1, C_2, C_3\}\{C_4, C_5\}$, and $\{C_1, C_2, C_3, C_4\}\{C_5\}$
  - Note $U = P_1 \cup P_2$ and $P_1 \cap P_2 = \emptyset$

    - Each covariate value (e.g. C_1) identifies a set of genomic positions having that value.
        - For cell-type specific covariates, the number of cell types having a covariate value at a particular genomic position is reported, if it is >0. This is used to weight the INSIGHT likelihood associated with C_i for a position according the fraction of cell types having class C_i in that position.
- Identify genomic associated with each coset
    - Generate a set of bit-masks that identify positions in the bit-mapped covariate database that are included in each co-set P_j. Keep a list of all coset pairs as the database is scanned. (covDB_t::bitfieldDB::vscanTarget_t)
    - Scan the entire covariate database For each position in the covariate database, identify fraction of position associated with each coset (rt.covdb.scanpatInit(), rt.covdb.scanpatGetNext())
    - If any coset is empty, delete that coset pair.
- Identify conditional Insight Log Likelihood for each partition as $LL_{INSIGHT}(Positions\ in\ P_1) + LL_{INSIGHT}(Positiosn\ in\ P_2)$
    - Calculate Insight2 scores (ρ) and data log(likelihood) for each coset in each coset pair
        - Use round robin allocation to initialized Insight2 processed. Log files for each of the N insight2 processes in ./insight2.log.N
        - (rt.insight.jobInit(); rt.insight.jobWait(); partitions[i].resultsRead())
        - Results for each coset saved in ./part/*
    - Estimate the information in a split as
      $$\delta I = -\Big(LL_{INSIGHT}(U) - \big(LL_{INSIGHT}(Positions\ in\ P_1) + LL_{INSIGHT}(Positions\ in\ P_2)\big)\Big)$$
    - When the logarithm base is 2, information is measured in ***bits***
    - Insight measures information in base e , a unit called ***nats*** . Conversion to bits is a multiplication.
- Sort partitions according to δI_C (main:: sortPartitions())
    - Save sorted partition information in ./covtree.outpart.txt
    - Identify the partition in C with the greatest δI as δI_C
    - Select the particular covariate $\bar{C}$ with the highest δI_C and split positions
    - Eliminate any partition that fails to meet quality control criteria. Possible control criteria include minimal number of positions, minimal information gain or minimal distance between values for ρ in each coset.
- Save the identified “best partition” as the selected binary split. If no split meets quality control indicate that recursion has terminated.

Split covariate values according to selected position and prepare for next iteration (main: bool processRecursion())

- Convert covariate cosets into two lines in the file ./covtree.out.
    - These become the input arguments to the next iteration of covtree.
- Create ./hi and ./lo subdirectories to contain the next iteration of results. ./hi contains the results from the first line in ./covtree.out, ./lo contains results from the second the second.
- If erecurse (internal recursion) is indicated, push 1 or 2 sets of arguments onto argument stack and reinitialize.
- Write ./covtree.done file as a semaphore to indicate to external job control logic that this iteration is complete.
    - If erecurse is 2, the covtree program will exit. It is expected that internal job control logic will generate two new instances of the covtree program, one for the ./hi result and one for the ./lo result and submit them to a queuing system. This improves parallel execution of the tree.
    - Before covtree exits, it writes a special like to the Insight2 instances that causes them to terminate normally. If they fail to do so, a signal is sent for them to exit ungracefully (insight.serverTerminate()).
    - Decision trees typically contain fewer than 200 nodes.

Notes

Covtree demos all use erecurse=0, that is, all recursion is accomplished via the argument stack and covtree does not exit until all recursion is complete. This would result in excessively long execution times when using the complete database, so the results generated from, the paper used erecurse=2, spawning up to 2 children from each covtree invocation. The control logic to synchronize these is not included in the Git repository.

The original likelihood calculation for a collection of N genomic positions was based on Insight ρ as

$$LL_{INSIGHT} = N(\rho \log(\rho) + (1-\rho)\log(1-\rho))$$

rather than the total InsightModel LL returned by Insight. Changes in model misspecification of total number of sites under selection rendered best-partition assessment ambiguous, so the actual model log(likelihood). This also enabled the identification of optimal partitions in which each child had similar values for ρ but differences in other parameters (η, γ). Several features reference the calculation of $LL_{INSIGHT}$ based on ρ. IN particular

- `-rhosplit - if present, split on rho entropy, rather than default of Insight2NLL.`
- `-hcmndinf [d] - If >0 use expected directed Information to determine ordering`
- `-implprho [d] - Alone or followed by 1 sets implicit-parent-Rho mode`

Those are not active in the reported results.

Approximately 3,400 lines of C++. No external library dependencies beyond STL and the butils headers developed for this software (fastfile, vecarray, stringer, mathplus) , and included with it.

--Brad Gulko