Naïve-Bayes greedy search for representative datasets.

We denote a subset of datasets whose signals can help us recover the full-stack state annotations as the representative set M^*. We applied a greedy search of datasets to form the representative set M^*, i.e. at each iteration, we search for one experiment to add to the representative set M^* such that the newly modified set of experiments can best recover the original full-stack state annotations. The initial representative set is empty. As re-computing chromatin state assignments for many different subsets of candidate experiments, *M*, based on ChromHMM's normal procedure would not be computationally practical, we instead used an approximation of chromatin state assignments with a Naïve Bayes approach. First, using the full-stack state annotations and the binarized genome-wide signals of 1032 datasets, we define the following properties:

- The state assignment at each 200bp segment, indexed i, along the genome is S_i.
- The prior probabilities of each state s is estimated as:
  $P(S_i = s) = \frac{\#\, positions\ where\ state\ s\ is\ assigned}{\#\, positions\ on\ the\ genome}$, for any position i
- The conditional probability of a dataset m being present (1) in each state s is estimated as:
  $P(m_i = 1 \mid S_i = s) = \frac{\#\, positions\ where\ mark\ m\ is\ present\ and\ state\ s\ is\ assigned}{\#\, positions\ where\ state\ s\ is\ assigned}$, for any position i.
  Then, $P(m_i = 0 \mid S_i = s) = 1 - P(m_i = 1 \mid S_i = s)$.

Given we are using a candidate representative set of datasets M, at each genomic position i, the binarized signals of a potential representative set is denoted $\overrightarrow{M_i}$, and the chromatin state assigned here is denoted S_i. We then calculate the posterior probabilities that this position is assigned to a particular state s as:

$$P(S_i = s \mid \overrightarrow{M_i}) = \frac{P(S_i = s) \times P(\overrightarrow{M_i} \mid S_i = s)}{P(\overrightarrow{M_i})} \propto P(S_i = s) \times \prod_{m \in M} P(m_i \mid S_i = s)$$

Position i is assigned to the state with highest posterior probability, i.e. $argmax_s\, P(S_i = s \mid \overrightarrow{M_i})$.
The greedy algorithm we applied to obtained a representative subset of datasets M^* is as follows:

- Step 0: The representative set M^* at time 0 is empty: $M_0^* = \emptyset$.
- Step 1: For each of the remaining set of datasets, add each one to the current representative set separately, and denote each of those sets a candidate set of representative datasets M.
- Step 2: Apply Naïve Bayes framework to obtain genome-wide state assignment using the candidate representative set M, denoting such annotation S_M.
- Step 3: Define the recovery score for each candidate set M as:
  $$Re(M) = \sum_{s=1}^{100} \frac{\#\, positions\ where\ state\ s\ is\ assigned\ correctly\ in\ S_M}{\#\, postions\ where\ state\ s\ is\ assigned\ in\ full\text{-}stack\ segmentation}$$
  and choose the candidate set with the highest recovery score, i.e. $argmax_M\, Re(M)$.

Repeat steps 1-3 to add one dataset to the representative set M^*at each iterative.

Asymptotic worst case time and memory usage of stacked model

Worst case compute time for each training iteration will grow linearly with the total length of the genome sampled per iteration. When the number of input tracks are sufficiently large, the worst-case compute time will grow linearly in the number of tracks. More specifically, let *M* be the number of input tracks (in our case, 1032), let *N* be the number of states (in our case, 100), let *L* be the length of the longest sequence (in our case, 5000, corresponding to one 1M-bp region of 200bp bins that we divided the genome into), let *S* be the number of sampled sequences per iteration (in our case, 300), and p the number of parallel processors (in our case, 6). For one iteration, the worst-case time on total compute usage is: $O(S(LN^2 + M(min(L, 2^M)N + L)))$.

This is the case since before running the forward-backward algorithm, ChromHMM pre-computes the emission probabilities for each unique combination of observed marks and state in the sample sequences (in our case, the 300 regions randomly sampled at each iteration). Determining the combination of marks' signals that each bin corresponds to requires a full pass on the sampled data, which is time $O(MSL)$. The number of observed combinations of marks per sequence is bounded by the smaller value between (1) the length of the sequence, L, and (2) the number of possible combinations with *M* inputs, which is 2^M. The time to compute the emission probability for a single combination of marks for every state is $O(MN)$ and time $O(SMN * min(L, 2^M))$ for every combination in each iteration (with S sample sequences of maximum length L). With pre-computed emissions, running the forward and backward algorithm on a sequence of length *L* takes $O(LN^2)$ time. This would need to be done on each of the *S* sampled sequences per iteration, thus taking $O(SLN^2)$ total. The update to the emission parameters after running the forward-backward algorithm on each sequence S can also be done in $O(SMN * min(L, 2^M))$ time. The default number of training iterations, which we used here is 200.

The worst case wall-time per training iteration is $O(((S(LN^2 + M(min(L, 2^M)N + L)))/p + N(M + N)S)$ assuming efficient parallelization. This is the case since the entire procedure is parallelized across p processors except for aggregating statistics on the $O(N^2)$ transition and $O(NM)$ emission parameters from each of the p processors, which takes $O(N(M + N)S)$ time.

The worst case memory usage of the procedure is $O(p(LN^2 + Mmin(L, 2^M)) + N(M + N)S)$ since for each of the p sequences being actively processed, the amount of memory to store the input is bounded by $O(Mmin(L, 2^M) + L)$ and the forward-backward procedure needs $O(LN^2)$ additional memory. Additionally, the output for each of the *S* sequences is stored before they are combined requiring $O(N(M + N))$ storage for each sequence.