SUPPLEMENT FOR:

Learning strong and weak signals in genomic sequence alignments to identify functional elements

James Taylor, Svitlana Tyekucheva, David C. King,
Ross C. Hardison, Webb Miller and Francesca Chiaromonte

Center for Comparative Genomics and Bioinformatics,
The Pennsylvania State University

Corresponding authors:

James Taylor
Center for Comparative Genomics and Bioinformatics
506 Wartik Bldg
Pennsylvania State University
University Park PA, 16802
Email: james@bx.psu.edu

Francesca Chiaromonte
Center for Comparative Genomics and Bioinformatics
505 Wartik Bldg
Pennsylvania State University
University Park PA, 16802
Ph: 814 865 7075
Email: chiaro@stat.psu.edu

Extended HKY+Gap model

To make inferences on ancestral base distributions, we must first introduce a model of nucleotide substitution for estimating the probability of a given substitution event over a given branch of the phylogenetic tree. We assume a continuous time Markov process in which a rate matrix Q specifies the instantaneous rate of each substitution event, and express the rates in Q through a smaller number of parameters. In particular, we use the parameterization provided by the HKY model of Hasegawa et al. (1985) consisting of equilibrium probabilities for each base (4 parameters; $\pi_{A,}\pi_{C}\pi_{G,}\pi_{T}$), and the ratio between the rates of transitions and transversions (κ). We extend this model to accommodate gaps as if they were a fifth nucleotide, introducing an additional equilibrium probability (π_{Gap}) and rate ratio (gaps to transversions σ), yielding the rate matrix:

$$Q = \begin{pmatrix} - & \pi_C & \kappa\pi_G & \pi_T & \sigma\pi_{Gap} \\ \pi_A & - & \pi_G & \kappa\pi_T & \sigma\pi_{Gap} \\ \kappa\pi_A & \pi_C & - & \pi_T & \sigma\pi_{Gap} \\ \pi_A & \kappa\pi_C & \pi_G & - & \sigma\pi_{Gap} \\ \sigma\pi_A & \sigma\pi_C & \sigma\pi_G & \sigma\pi_T & - \end{pmatrix}$$

We estimate the parameters in Q using the Expectation Maximization algorithm implemented in the PHAST software package (Siepel and Haussler, 2004). For the applications presented in this paper, we fix the tree topology as that of Margulies et al. (2006), and run the estimation on a sample of genome-wide alignments of human, chimpanzee, macaque, mouse, rat, cow, and dog (for RP and hypersensitive sites), or human, mouse, opossum, chicken, frog, zebrafish, and pufferfish (for conserved elements with developmental enhancer activity).

Ancestral base distribution inference

Given an alignment column $x = (x_1,\ldots x_m)$, the posterior distribution for the base in the common ancestor of the m species is

$$\Pr(y \mid x) = \frac{\Pr(x \mid y)\pi_y}{\sum_{z\in\{A,C,G,T,Gap\}} \Pr(x \mid z)\pi_z} \qquad \text{(S.1)}$$

Felsenstein's algorithm evaluates the likelihood $\Pr(x \mid y)$ in Equation (S.1) recursively proceeding bottom-up along the phylogenetic tree through a series of "triangulations". For a generic stage, let y_0, y_1 and y_2 be, respectively, the position for the ancestor currently under consideration (0), and its two immediate descendants (1 and 2). The basic recursive relation is

$$\Pr(x(0) \mid y_0) = \sum_{y_1} \Pr(x(1) \mid y_1) \Pi_{y_0 \to y_1}(\tau(0,1)) \times \sum_{y_2} \Pr(x(2) \mid y_2) \Pi_{y_0 \to y_2}(\tau(0,2))$$

where $x(\mathcal{A})$ indicates the subset of x corresponding to observed species descending from $\mathcal{A}$, $\tau(\mathcal{A},\mathcal{B})$ the length of the branch linking $\mathcal{A}$ and $\mathcal{B}$, and $\Pi_{y_{\mathcal{A}} \to y_{\mathcal{B}}}(\tau(\mathcal{A},\mathcal{B}))$ the corresponding transition probability obtained through

$$\Pi(\tau) = \exp\{-Q\tau\} = \sum_{j=0}^{\infty} \frac{(-Q\tau)^j}{j!} \qquad \text{(S.2)}$$

The Q in Equation (S.2) and the π's in Equation (S.1) are, respectively, the rate matrix and equilibrium distribution of the of the HKY+Gap substitution model described above.

Clustering based on proximity and entropy

Ancestral base distributions are points in the 5D simplex. We group them using a novel agglomerative clustering algorithm that combines entropy and spatial proximity in the simplex. At each iteration, a merger is chosen among a set of candidates as to maximize entropy of the resulting partition. Let G indicate the current partition in groups g each containing a fraction $f_g = n_g/n$ of the training column occurrences (that is, n_g is the sum of occurrences of all alignment columns that correspond to ancestral distributions in g, and n is the overall number of alignment columns in the training data). Also, let C indicate a set of candidate mergers c, and $G(c)$ the partition in groups $g(c)$ (each containing a fraction $f_{g(c)}$ of the occurrences) resulting from merger c. We select the merger

$$c^* \text{ such that } H(G(c^*)) = -\sum_{g \in G(c^*)} f_{g(c^*)} \log(f_{g(c^*)}) = \max_{c \in C}$$

Because $G(c)$ is "nested" in G, the entropy of the former coincides with the mutual information between the two so that, at each iteration, selecting a merger to maximize entropy is the same as selecting a merger to retain maximal information relative to the current partition. Because it uses entropy, this algorithm tends to create clusters of similar size, located depending on the frequency of occurrences in the simplex.

Proximity is used as a constraint; by limiting the set of candidates C in each iteration to mergers involving "neighboring clusters", we ensure that clusters remain spatially contiguous. This can be implemented in several ways, as to give stronger of weaker roles to proximity vis a vis the entropy maximization. For the applications presented in this paper, we let a merger $c = \{g_1, g_2\}$ be a candidate if

$$d(g_1, g_2) = \min_{g \in G} d(g_1, g) \text{ or } \min_{g \in G} d(g, g_2)$$

Although other choices are possible, we use Euclidian distance and a centroid linkage, i.e. we define the distance between two clusters as the Euclidean distance between their centroids. Also, for the applications presented here we implement a pre-clustering step: before starting the agglomeration we merge ancestral reconstructions corresponding to alignment columns that occur less than ν times (e.g. 5) in the training data to the closest ones that occur at least ν times.

Evaluation of encodings through cross validation

To evaluate the classification performance of an encoding during the iterative search, we use k-repeated h-fold cross validation. The training data is partitioned at random into h (e.g. 10) folds, a fold is withheld, and two variable order Markov models are estimated with the remaining positive and negative data. The estimated models are used to produce log-odds scores for all the data (including the withheld fold). If the sets of scores for positive and negative data used in training overlap, withheld data is classified into positive and negative based on the sign of their scores. If the sets do not overlap, the withheld data is classified as positive if their score is larger than the minimum score of

the positive data, as negative if it is smaller than the maximum score of the negative data, and as "unclassifiable" if it falls in between. This yields counts of correctly classified, erroneously classified, and unclassifiable elements in the withheld fold. The process is repeated for the h folds, and for k (e.g. 10) random partitions of the data. Counts are averaged in correct classification (success), erroneous classification, and unclassifiable rates associated with the alphabet.

Unlike the success rates used to evaluate encodings during the search, the ones reported in outcome of ESPERR applications, i.e. the success rates obtained on optimal encodings, are recomputed with leave-one-out cross validation for stability (instead of withholding folds, the data elements are withheld one at a time).

Sampling of candidate encodings and heuristics for the iterative search

Our search generates candidate encodings, accepts the best based on a figure of merit (FOM), and repeats until a good encoding is found. The FOM is the cross validation success rate described above, and does not include "unclassifiable" elements. At each stage, candidates are generated from the current encoding by either merging two symbols (groups) or extracting an atom from one of the symbols. When the current encoding is large, many candidates will perform close to (a poor) best. Thus we evaluate only a random sampling, e.g. γ=50 mergers and η=20 extractions, which reduces computations while still producing reasonable moves with high probability. As the current encoding shrinks, γ represents a larger fraction of the possible mergers, and η random extractions continue to afford a degree of reversibility to the search.

Large encodings require more parameters, are more susceptible to over-fitting and thus score more elements in the unclassifiable range, reducing the FOM. Consequently, the search strongly prefers small encodings, and it is possible that evaluating single atom extractions will not be enough to by-pass local optima. We overcome this problem with a heuristic: if the FOM does not increase over w (e.g. 20) consecutive iterations, we consider only extractions for e (e.g. 5) consecutive steps, which allows us to move out of local optima through poorer performing, larger encodings.

Even with this heuristic, it is still possible for the search to make bad moves which then take a long time to be reversed. To recover efficiency, we add a "restarting"

heuristic: if we proceed for *r* (e.g. 50) iterations without reaching an encoding better than the best seen so far, we restart the search at that best encoding. Termination is similar but extends to a much larger number of iterations – we stop if we go for 1,000 iterations without reaching an encoding better than the best seen so far, and adopt that best encoding as the final one.

Pseudo-code for the randomized search algorithm

The search is initialized using some mapping, either a one-to-one mapping of the training data symbols (e.g. all alignment columns) or the result of another encoding selection procedure (e.g. the clustering based on ancestral base distributions). After each iteration, this will be replaced with the best mapping found in that iteration.

```
mapping = initialize_mapping()
```

We keep track of the best mapping seen, and its figure of merit. When the search terminates this best mapping corresponds to the final encoding.

```
best_merit_overall = -Inf
best_mapping_overall = None
```

The search iterates until it has performed 1,000 iterations without any improvement over the best mapping seen.

```
while steps_since_best < 1,000:
```

Within each iteration, we keep track of the best candidate mapping found.

```
    best_merit = -Inf
    best_mapping = None
```

The first set of candidate mappings is created by merging symbols in the current encoding. We consider a random sample of γ such candidates. For practical reasons we set a lower bound (e.g. 5) on the encoding size and skip this step if the encoding is already too small.

```
    if symbol_count > minimum_alphabet_size:
```

Sample γ pairs from all possible pairs of symbols that could be collapsed.

```
        for pair in sample( all_collapsible_pairs( mapping ), γ ):
```

Generate a new mapping in which that pair of symbols are merged

```
            new_mapping = collapse( current_mapping, pair )
```

Evaluate the figure of merit when this mapping is applied to the training data. If it is the best so far for this iteration, save it.

```
merit = calc_merit( new_mapping )
if merit > best_merit:
   best_merit = merit
   best_mapping = new_mapping
```

The second set of candidates is created by extracting atoms which are currently grouped with other symbols. We consider a random sample of η such candidates. Again for practical reasons we only break out seeds which occur more than 10 times in the training data, since they will not comprise any context that can be incorporated in the model (see VOMM estimation).

```
for atom in sample( expandable_atoms( mapping ), η ):
```

Generate a new mapping with that atom separated.

```
   new_mapping = expand( mapping, atom )
```

Evaluate the figure-of-merit when this mapping is applied to the training data. If it is the best so far for this iteration, save it.

```
   merit = calc_merit( new_mapping )
   if merit > best_merit:
      best_merit = merit
      best_mapping = new_mapping
```

We accept the best mapping from either the collapse or expand steps as the new mapping for the next iteration

```
mapping = best_mapping
```

When the new mapping is better that the best seen so far, we save it and reset the counters used to trigger the two heuristics and termination.

```
if best_merit > best_merit_overall:
   best_merit_overall = best_merit
   best_mapping_overall = best_mapping
   steps_since_best = 0
   steps_since_restart = 0
   steps_since_forced_expansion = 0
```

We now check if the "restarting" heuristic should be triggered. If we have gone r iterations without an improvement over the best mapping, we restart from that mapping and reset the counters for the heuristics.

```
if steps_since_restart >= r:
   steps_since_restart = 0
   steps_since_forced_expansion = 0
   mapping = best_mapping_overall
```

Next we check if the "forced expansion" heuristic should be triggered. If we have gone w iterations without improvement over the best mapping, we force e consecutive expansion steps. These expansions are part of a single "iteration" and do not affect the counters (the expansion procedure is otherwise identical to that above).

```
if steps_since_forced_expansion > w:
   steps_since_forced_expansion = 0
   for i from 0 to e:
      best_merit = 0
      best_mapping = None
      for atom in sample( expandable_atoms( mapping ), η ):
         new_mapping = expand( mapping, atom )
         merit = calc_merit( new_mapping )
            if merit > best_merit:
            best_merit = merit
            best_mapping = new_mapping
            mapping = best_mapping
```

Finally we increment the counters that keep track of when each heuristic is triggered and when the search terminates.

```
steps_since_best += 1
steps_since_restart += 1
steps_since_forced_expansion += 1
```

Variable order Markov models and their estimation

A Markov model of fixed order T on a state space S is usually represented through a $\#(S)^T$ by $\#(S)$ transition probability matrix, whose entries $p(s|\, s_{-1}\ldots s_{-T})$ express the chances of s conditional to the symbols in the T preceding positions. An alternative and more intuitive way of representing Markov models is through a tree structure; each node in the tree

correspond to a context of a given length, say *a,b* of length 2, and contains transition probabilities $p(s|b,a)$, s in S. The children of such node correspond to contexts extended forward by one symbol, say *a,b,c*, and contain transition probabilities $p(s|c,b,a)$, s in S . A tree comprising all contexts up to length T contains in its leaf nodes all the transition probabilities required to specify a Markov model of fixed order T. A variable order Markov model (VOMM) of maximal order T can be thought of as a "pruned" version of such a tree, where a reduced number of leaf nodes correspond to contexts of variable lengths with distinct transition probabilities.

Fitting a VOMM on training data consists of extending contexts, and estimating the corresponding transition probabilities. We extend contexts using a pruning criterion; considering each order t from 0 to T, we augment the tree to include a node for each context $s_{-t}\ldots s_{-1}$ that occurs more than p (e.g. 10) times in both the positive and the negative training sets. While this criterion is naïve compared to other VOMM pruning strategies, it does not require the maximal model (where all contexts are considered) to be built before pruning, and thus allows quicker model fitting. For each node included in the tree, we then need to compute the transition probabilities $p(s|\, s_{-1}\ldots s_{-t})$, s in S. Of course a node may not have a full set of children, and there may even be extended contexts $s_{-t}\ldots s_{-1}$, s that never occur in the data. To produce non-zero estimates for the corresponding probabilities, we use a "discount" smoothing rule, which redistributes a small amount of mass d (e.g. 0.01) through the formula:

$$p(s \mid s_{-1}\ldots s_{-t}) = (1-d)\frac{\#(s \mid s_{-1}\ldots s_{-t})}{\sum_{\tilde{s}\in S}\#(\tilde{s} \mid s_{-1}\ldots s_{-t})} + d\ p(s \mid s_{-1}\ldots s_{-(t-1)}) \qquad s \in S$$

where $\#(.|\, s_{-1}\ldots s_{-t})$ indicates number of occurrences after $s_{-t}\ldots s_{-1}$ (in other words, the rule reallocates d mass relative to the distribution of the parent context $s_{-(t-1)}\ldots s_{-1}$). For order zero (empty context) we set d=0.

Note that the maximal order is a hard limit on the size of a VOMM, since contexts can not extend beyond T. Pruning also limits the size of the model, as it determines how many transition probabilities need to be estimated. Preliminary investigations showed

that our fits are robust to changes in p and d, at least for relatively small values of these parameters.

Log-odds classification

For classification, we fit two variable order Markov models on the positive and negative training sets, as described above. Any training or independent alignment segment, say $a = (a_1 \ldots a_n)$ comprising n columns, is then scored with the equation

$$\ell(a) = \sum_{i=1\ldots n} \log\left(\frac{p_{POS}(a_i \mid a_{POS}^{(i-)})}{p_{NEG}(a_i \mid a_{NEG}^{(i-)})} \right)$$

where $a^{(i-)}_{POS}$ and $a^{(i-)}_{NEG}$ represent the relevant contexts (symbols in position i-1, i-2…) under the positive and negative model. $\ell(a)$ is positive if the patterns in a resemble those characteristic of the positive training data, and negative if the resemblance is to the negative training data, so the segment can be classified by the sign of its score.

Receiver Operating Characteristic (ROC) Curves

Let $g(x)$ be a generic score for entities x with known labels $l(x)$=POS or NEG, and consider a classification rule that predicts labels based on a threshold y as $pl(x)$=POS if $g(x) > y$ (NEG if $g(x) \leq y$). Correspondingly, define true positive/negatives and false positive/negatives associated with y as:

$$TP_y = \{x: pl(x) = l(x) = POS\} \quad , \quad TN_y = \{x: pl(x) = l(x) = NEG\}$$
$$FP_y = \{x: pl(x) = POS, l(x) = NEG\} \quad , \quad FN_y = \{x: pl(x) = NEG, l(x) = POS\}$$

The sensitivity and specificity associated with y are then given by:

$$Sp_y = \frac{\#\{TN_y\}}{\#\{TN_y\}+\#\{FP_y\}} \quad , \quad Sn_y = \frac{\#\{TP_y\}}{\#\{TP_y\}+\#\{FN_y\}}$$

expressing, respectively, the share of negative predictions that are true negatives, and the share of positive predictions that are true positives. The ROC Curve plots the locus $\{Sn_y; 1\text{-}Sp_y\}$ as the threshold y varies, capturing the underlying trade-off between sensitivity and specificity of the classifier. ROC curves can be used to compare performance for different classifiers over ranges of thresholds – as sensitivity and specificity range, in trade-off, between 0 and 1.