Princeton University – Computer Science Senior Independent Work

An Evaluation and Extension of Sequence Conservation Methods for Protein Functional Site Prediction

Josh Chen

Advisor: Mona Singh

January 7, 2014

Abstract

Scoring the sequence conservation of the amino acid sites of a protein is a powerful way to gain insight into protein function. INTREPID and Rate4Site (ConSurf) are two well-used algorithms for scoring sequence conservation, but there does not exist an unbiased evaluation of their prediction performance or a thorough specification of the latter's implementation. I fill in this gap by carefully detailing the two algorithms, methodically evaluating their relative performance, and developing a Python package I call 'ConsEval' to support the efficient implementation, execution, and evaluation of these and similar protein functional site scorers. Some notable results from my evaluation step are data to contradict the published claim that INTREPID significantly outperforms similar scorers; tests of parameter settings that recommend improved parameters for a third sequence conservation scoring method, the Jensen-Shannon scorer from Capra and Singh (2007); new justification for evaluating scoring methods using precision-recall curves; and a number of suggestions for future work.

1. Introduction

Efficiently decoding the function of sequenced proteins is one of biggest challenges in molecular biology. Understanding protein function aids the biologist in everything from analyzing the results of biological experiments, to developing targeted drugs for diseases, to evaluating theories on the molecular mechanisms of life. Typically, due to the massive amount of protein data needed to be analyzed and the expense of experimentally determining protein function, emphasis has been placed on using computational methods to infer function [25, 13]. One class of these methods focuses on computationally identifying the functional residues of a protein, based on the observation that a protein's overall function can usually be elicited from its limited number of functional sites [4]. It should be noted that in addition to aiding greatly in the general problem of decoding overall function, identifying functional sites is also interesting in itself for helping pinpoint protein regions of specific modeling or experimental interest such as catalytic sites, ligand-binding sites, protein-protein interfaces, and specificity determinants.

The simplest type of data that can be used to identify functional sites is the unannotated amino acid sequence of a protein. Using only a protein's sequence, one can perform a BLAST search for sequentially similar homologs, run a multiple sequence alignment tool like ClustalW to match similar regions, and construct an evolutionary tree of likely ancestral paths via tools like PhyML or MrBayes; this gives rise to sequence conservation methods that sift for sites well-preserved through evolution. Methods utilizing additional information abound: One might look for amino acids especially unique or especially conserved in structure [20], filter for regions of high relative solvent accessibility [35], make predictions by combining multiple types of features and annotations [12, 35], or else. These methods are naturally more accurate, identifying functional sites with better prediction performance on experimental data than methods using sequence data alone [17, 34]. However, these methods are not generalizable to the substantial fraction of interesting proteins without solved structures [27], and additionally, it has been found that sequence conservation is one of the strongest predictors of amino acid function [35]. For these reasons, I focus on sequence-based site inference methods, and sequence conservation methods in particular, in this paper.

In this paper, I start by briefly presenting the specific technical problem of identifying functional sites by scoring sequence site conservation (Section 2). I then carefully detail the approaches and implementations of the INTREPID algorithm [27] and the empirical version of the Rate4Site algorithm (R4S-EB) [18] (Section 3); R4S-EB and INTREPID lie at the heart of the well-used ConSurf [11, 1, 5] and INTREPID [26] Web servers, respectively. Third, I evaluate INTREPID and R4S-EB's prediction performance by running new implementations of their methods against two labeled test sets from Capra and Singh 2007 [2] (Section 4). This evaluation leads me to make several important observations: I detail the empirical differences between INTREPID and R4S-EB's prediction performance, I discover that the simple Jensen-Shannon divergence method can be improved by tuning its parameter to $\lambda = 0.1$, I argue that it's more meaningful to evaluate scorers using precision-recall than using the currently more common ROC, and I suggest and test some modest adjustments to INTREPID and R4S-EB's algorithms. Perhaps most importantly, I develop and present a software package enabling the rapid implementation of functional site inference methods, the definition of easily replicable evaluators for these methods, and the parallelized batch execution of implemented scorers using any array of parameter settings (Section 5). This package, provided along with my own implementations of INTREPID and Rate4Site, is available on the Web at http://github.com/jwayne/iw-conservation/.

2. Problem

Identifying the functional sites of a protein $\boldsymbol{x} = \{x_1, \ldots, x_L\}$ involves determining, for each amino acid x_i, a score s_i for how strongly x_i is predicted to be directly involved in biological function. Sequence conservation methods do this by applying the theoretical intuition that biologically important amino acids should be preserved across evolutionary homologs [22]; these methods score each x_i for how well-preserved, according to some measure approximating amino acid function, x_i is in its column X_i of a multiple sequence alignment $\boldsymbol{X} = \{X_1, \ldots, X_L\}$. I note that there are many broad definitions of having a site be involved in function. Functional sites might be active sites of interaction with enzymes, target sites of post-translational modifications, target sites for ligand (substrate or nucleic acid) binding, etc. [6]. It is clear, however, that catalytic sites and ligand-binding sites ought to be marked as functional. This target has been optimized for in multiple past evaluations [2, 35, 27], and it is the objective I use as well.

There appear to be two main approaches to modeling per-site sequence conservation. Position-by-position heuristic methods assemble a simple score measuring the deviance among amino acids in each column; these methods often make use of basic measures from information theory. INTREPID falls into this category, as do many earlier and simpler methods like the several scorers centered on the Jensen-Shannon divergence, on relative entropy, and on Shannon entropy evaluated by Capra and Singh [2]. On the other hand, Bayesian and maximum likelihood methods define a probabilistic model for the evolution of amino acids in an (often precalculated) evolutionary tree. This approach is considerably more complex, but Bayesian methods enable the intuitive description of variable amino acid evolution rates at different sites in a protein as part of an elegant probabilistic model for that protein's evolution. R4S-EB, as an empirical Bayes method,[1] fits into this category. (A related earlier algorithm, the maximum likelihood version of Rate4Site at [24], should be mentioned alongside the Rate4Site's empirical Bayes version as well.) I detail the models and implementation behind INTREPID and R4S-EB methods in the next section.

3. Key Algorithms

3.1. INTREPID: Maximization of divergence scores over evolutionary subtrees

The INTREPID [27] algorithm is quite simple to implement. It is motivated by a minor weakness its authors discovered in earlier heuristic sequence methods: INTREPID's authors observed that earlier heuristic methods (e.g., [2]) measured for the conservation of amino acids across *all* sequences in a column, even as a properly conserved functional site might not be so conserved across every such sequence. Hence, INTREPID adjusts for this by using information from an evolutionary tree to generalize those earlier simple "global" scoring approaches.

To score a site i, INTREPID computes the Jensen-Shannon divergence $J(S,i)$ of its column for each subtree S along the path from the given phylogenetic tree's root to the leaf node of the protein

[1]Empirical Bayes methods, just like fully Bayesian models, perform Bayesian inference to find the expectation of the posterior $E[\theta|X,\alpha]$. However, they also further optimize hyperparameters α that are considered fixed in standard Bayesian methods, by instead using $\hat{\alpha} = \arg\max_{\alpha} P(X|\alpha)$. See [3] for an excellent discussion of empirical Bayes, fully Bayesian, and maximum likelihood methods.

sequence of interest. These divergences are then, for each subtree S, averaged across all sites in the sequence to obtain the mean subtree divergences $\overline{J(S)}$. Finally, the conservation score s_i is computed as the maximum of the difference $J(S,i) - \overline{J(S)}$ over all subtrees:

$$s_i = \max_S (J(S,i) - \overline{J(S)}) \tag{1}$$

The Jensen-Shannon divergence (JSD) is used over other global divergence scoring measures like relative entropy and Shannon entropy because it was previously found to outperform those other measures [2]. As applied to scoring sequence conservation, the JSD can be understood, at a high level, as a measurement of the divergence between the frequency distribution p of amino acids within the alignment column of interest and a given background amino acid distribution π. Its formula is:

$$J(i) = \lambda RE_{p,\phi} + (1-\lambda) RE_{\pi,\phi} \tag{2}$$

λ is a parameter called the prior weight, $\phi = \lambda p + (1-\lambda)\pi$ is a weighted sum of the distributions p and π, and $RE_{p,q}$ denotes the relative entropy between two distributions $RE_{p,q} = \sum_a \frac{p(a)}{q(a)} p(a)$. The INTREPID authors set the prior weight $\lambda = 0.5$ to match a previously recommended setting [2].

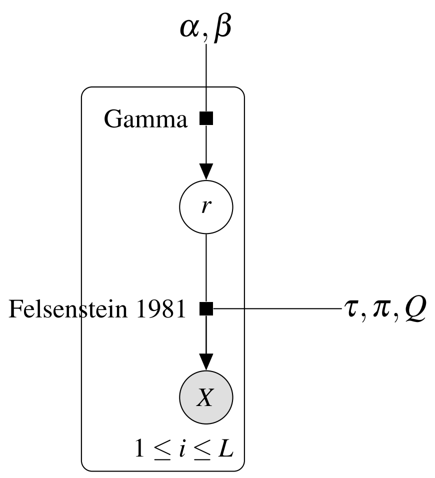
The JSD ranges in $[0,1]$ for any inputs, and so INTREPID scores range in $[-1,1]$. Higher scores indicate higher predicted levels of conservation.

3.2. R4S-EB: Empirical Bayes inference on a per-site variable-rate evolutionary model

The R4S-EB model is much more complex to implement, but it also has an intuitive motivation. In most phylogenetic tree builders' likelihood models, all sites in a protein sequence are assumed to evolve at the same rate. This is biologically unrealistic [31, 32], and so given an alignment and an evolutionary tree, R4S-EB [18] produces a variable relative rate of amino acid evolution $r_i \in [0,20]$ for each column X_i.[2] This rate specifically represents, for any parent and child node pair A, B in the input tree, an adjustment to the evolutionary distance separating sites in A, B as defined by the tree: Instead of being uniformly separated by the branch length t_{AB}, aligned amino acids

[2]The maximum rate is set as 20 in [18] to avoid spuriously high rates of evolution. When I implemented my version of R4S-EB, I confirmed this maximum to be quite useful in preventing naturally impossible rates from being inferred.

Figure 1: Probabilistic model for R4S-EB. For each site, the conservation rate r is selected from a Gamma prior with hyperparameters α, β and the column's amino acids X are selected based on Felsenstein's 1981 likelihood model for phylogenetic trees [9] marginalized over internal nodes. Additionally, τ is the provided tree (branch lengths and topology), π is a background distribution of amino acids, and Q is an instantaneous rate matrix for amino acid substitution.



at sites i in sequences A, B are modeled as being separated by an evolutionary distance of $r_i t_{AB}$. (Note that the average r_i must be 1.) This model maps directly onto the sequence conservation problem of predicting per-site levels of amino acid conservation: Lesser r_i imply stronger sequence conservation. One might obtain conservation scores s_i by negating and then normalizing the rates r_i over each sequence.

Figure 1 shows a plate diagram describing the probabilistic model for these rates.[3] Under (standard) fully Bayesian inference, this diagram implies the following equation for each site's r:

$$r = E[r|X; \tau, \pi, Q, \alpha, \beta] = \int_r r \frac{P(X|r; \tau, \pi, Q)\, P(r; \alpha, \beta)}{P(X; \tau, \pi, Q, \alpha, \beta)} \tag{3}$$

R4S-EB uses empirical Bayes estimation to set α, substituting $\hat{\alpha}$ for α in this formula. I leave the detail of this estimation process $\hat{\alpha}$ for Section 3.2.3. Here, I detail this plate diagram by describing the distributions for the prior $P(r; \alpha, \beta)$ and the likelihood $P(X|r; \tau, \pi, Q)$ in turn.

For the prior on r, each site's rate r is selected independently from a gamma distribution $P(r; \alpha, \beta) \propto r^{\alpha-1} e^{-\beta r}$. The motivation for using this distribution is cited as the use of the gamma to model variable evolutionary rates among sites in Yang 1993 [31]; Yang credits Nei and Gojobori 1986 [21] who then reference Uzzell and Corbin's original 1971 work [29] recommending the negative binomial distribution[4] for modeling the number of mutation fixations at each protein site. Note that the model presented here assumes that rates at different sites are mutually independent

[3]I omit the subscripts i as in [23]; this signifies in that the model applies to all sites.

[4]The negative binomial distribution is often considered a discrete approximation to the gamma.

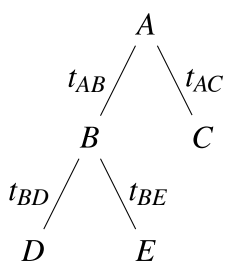


Figure 2: Example phylogenetic tree with observed leaves C, D, E and unknown internal nodes A, B. One can calculate Felsenstein's likelihood for this tree by performing a postorder traversal, marginalizing out nodes A, B.

given the hyperparameters α, β. Since rates are defined to have mean 1, R4S-EB lets $\beta = \alpha$ so that $E[r] = \frac{\alpha}{\beta} = 1$. The value of α (again, left up to empirical Bayes estimation) then determines the gamma's variance $Var[r] = \frac{\alpha}{\beta^2} = \frac{1}{\alpha}$.

As for the distribution of the likelihood $P(X|r;\tau,\pi,Q)$, R4S-EB applies Felsenstein's well-used 1981 model for computing the likelihood of a phylogenetic tree of DNA sequences. Under this model [9], given a tree τ with predetermined topology and branch lengths, the likelihood of observing some assignment of nucleotides to the tree's nodes is defined to be the background frequency π_{root} of the nucleotide at the tree's root multiplied by the product of each substitution probability along every branch AB. (This second component is $\prod_{AB} P_{ab}(t_{AB}) = (e^{Qt_{AB}})_{ab}$ where a and b are the nucleotides at nodes A and B, t_{AB} is the corresponding branch length provided as part of τ, and Q is the instantaneous rate matrix for nucleotide substitution.) Since all leaves are observed and all internal nodes are unknown in phylogenetic trees, this likelihood is marginalized over all possible identities of the internal nodes. As an example, the likelihood of the tree in Figure 2 is:

$$P(C=c, D=d, E=e; \tau, \pi, Q) = \sum_a \pi_a P_{ac}(t_{AC}) \sum_b P_{ab}(t_{AB}) P_{bd}(t_{BD}) P_{be}(t_{BE}) \quad (4)$$

In R4S-EB, the nucleotide substitution model is directly exchanged for a model of amino acid substitution. Furthermore, as dictated by R4S-EB's model of site-specific rates, branch lengths t are adjusted to be rt. In the likelihood's final incarnation for R4S-EB, π and Q describe amino acid substitution and the $P(t)$ become $P(rt)$.

The R4S-EB authors finish describing their model by specifying that they perform inference by discretizing the gamma prior on r into $K = 16$ bins. Computing the posterior Bayes estimate from Equation 3 then becomes the following sum over all K settings of r_k:

$$r = E[r|X;\tau,\pi,Q,\alpha,\beta] \approx \frac{\sum_{k=1}^{K} r_k P(X|r_k;\tau,\pi,Q)\, P(r_k;\alpha,\beta)}{\sum_{k=1}^{K} P(X|r_k;\tau,\pi,Q)\, P(r_k;\alpha,\beta)} \tag{5}$$

This sum is straightforward, but the binning and setting of the r_k is not. It turns out that the discretization of the gamma distribution should be done so that each bin has equal probability mass under the gamma. This was not obvious to me: I had to find [32] to learn that one should use equal probability mass binning, and then I had to reason that r_k should be set so that $E[r_k] = E[r]$. This implies a somewhat convoluted expression for setting the r_k representing each bin, as we need $E[r_k] = \sum_{k=1}^{K} r_k P(\text{bin } k) = \frac{1}{K}\sum_{k=1}^{K} r_k$ to equal $E[r] = \frac{\alpha}{\beta}$.

I was unable to analytically perform the derivation, but the source code for R4S-EB [18] [5] sets:

$$r_k = K\frac{\alpha}{\beta}\left(\gamma(\alpha+1,\beta b_k) - \gamma(\alpha+1,\beta a_k)\right) \tag{6}$$

Here, $a_k = \gamma^{-1}(\alpha, \frac{k-1}{K})$ is bin k's lower bound, $b_k = \gamma^{-1}(\alpha, \frac{k}{K})$ is bin k's upper bound, $\gamma(a,x) = \int_0^x t^{a-1}e^{-t}dt$ is the lower incomplete gamma function [30], and $x = \gamma^{-1}(a,y)$ is the inverse of $y = \gamma(a,x)$. Setting r_k in this way ends up producing r_k whose means numerically equal $\frac{\alpha}{\beta}$. I leave full derivation to future work.

This all said, there are still many further difficult details left undefined. The R4S-EB authors do not specify (1) how to actually obtain π and Q, (2) how to compute the matrix exponential $P(t) = e^{Qt}$, (3) how to efficiently compute Equation 4 over a tree after π, Q, and a method to compute P are obtained, and (4) how to compute an empirical Bayes estimate for the hyperparameter α. For (3), I refer the reader to Schabauer *et al.*'s explanation [28][6] of how marginalize out internal nodes by performing Felsenstein's postorder tree traversal [9] and multiplying probability matrices at each node. On the other hand, I could not find obvious references for (1), (2), or (4). I thus detail my own implementation of these steps in Sections 3.2.1 to 3.2.3.

3.2.1. π and Q can be obtained from several amino acid substitution models [15, 16] provided in the PAML format. Any model of amino acid substitution provides a set of background frequencies π along with a set of parameters describing the likelihood of every pairwise amino acid

[5] *cf.* src/phylogeny/gammaUtilities.cpp::the_avarage_r_in_category_between_a_and_b
[6] *cf.* section II.

substitution. Felsenstein's tree likelihood computation is particular about the format of the pairwise substitution parameters, however: To calculate the substitution probabilities $P(t) = e^{Qt}\ \forall t \geq 0$, an instantaneous rate matrix Q is needed. Unfortunately, Q cannot be obtained from the widely distributed likelihood ratio score versions of available substitution models, and what appears to be a standard method of computing Q via the eigen-decomposition of the probability matrix versions of these models [14] has been criticized as imprecise in [15]. Fortunately, well-tested and nearly directly-provided versions of Q can be found in [15, 16].[7]

These well-tested versions are published as files formatted for use by the phylogenetic analysis package PAML [33]. Briefly, files in this format provide π along with a lower triangular matrix S of "symmetric exchangeability parameters" without its diagonals filled in. The full matrix for S can be obtained by filling out the upper triangle and then filling out diagonal entries so that, when S is multiplied by $\Pi = \mathrm{diag}(\pi)$, the rows of $S\Pi$ sum to 0 [15]. Q is then obtained via $Q = S\Pi$. Note that this format requires S to be provided such that the row sums of Q without its diagonals are each 1; this puts P in time units of 1 expected mutation per site.[8] An snapshot of a substitution model saved in this format is shown in the Appendix.

3.2.2. The matrix exponential $P(t) = e^{Qt}$ can be efficiently computed via the repeated decomposition of Q.

Using Taylor series, the matrix exponential can be written as the following sum [19]:

$$e^{Qt} = \sum_{n=1}^{\infty} \frac{(Qt)^n}{n!} \tag{7}$$

This sum cannot be easily computed for general Q, but notably, $Q = S\Pi$ where S and Π are symmetric. This suggests letting $A = \Pi^{\frac{1}{2}} S \Pi^{\frac{1}{2}}$ [28], giving:

$$e^{Qt} = \sum_{n=1}^{\infty} \frac{(S\Pi t)^n}{n!} = \Pi^{-\frac{1}{2}} \sum_{n=1}^{\infty} \frac{(At)^n}{n!} \Pi^{\frac{1}{2}} = \Pi^{-\frac{1}{2}} e^{At} \Pi^{\frac{1}{2}} \tag{8}$$

The final expression here involves a well-conditioned matrix exponential e^{At} due to At being symmetric [19, 28]. Decomposing A into its right column eigenvectors U and eigenvalues $\Lambda =$

[7] I include the files for these versions in my software package described in Section 5; they directly are taken from the sources [15, 16].

[8] For example, using Kosiol and Goldman's Q for the Dayhoff substitution model with $t = 1$, one ought to obtain the PAM-100 matrix when computing $P = e^{Q*1}$.

diag$(\lambda_1, ..., \lambda_{20})$, we have $U^T = U^{-1}$ and thus $A = U\Lambda U^T = U\Lambda U^{-1}$. As a result:

$$e^{At} = \sum_{n=1}^{\infty} \frac{(At)^n}{n!} = U \sum_{n=1}^{\infty} \frac{(\Lambda t)^n}{n!} U^T = U e^{\Lambda t} U^T = U \cdot \text{diag}(e^{\lambda_1 t}, ..., e^{\lambda_{20} t}) \cdot U^T \tag{9}$$

$$e^{Qt} = \Pi^{-\frac{1}{2}} U \cdot \text{diag}(e^{\lambda_1 t}, ..., e^{\lambda_{20} t}) \cdot U^T \Pi^{\frac{1}{2}} \tag{10}$$

Equation 10 is a simple matrix multiplication. I note, however, that it is crucial to optimize this computation: e^{Qrt} must be computed for each branch and for each value of r that one wishes to find $P(rt)$ for.[9] Since I perform Bayesian inference by discretizing the prior for r as specified by the R4S-EB authors,[10] I make almost no optimization, pre-computing only $Y_L = \Pi^{-\frac{1}{2}} U$ and $Y_R = U^T \Pi^{\frac{1}{2}}$ to give two full 20×20 matrix multiplications per computation of $P(rt) = Y_L e^{\Lambda rt} Y_R$. [28] discusses an optimization leveraging the symmetry of Π and e^{Λ} that would cut my runtime in half.

3.2.3. The empirical Bayes estimate $\hat{\alpha}$ of α can be computed using the expectation-maximization (EM) algorithm. The empirical Bayes estimate $\hat{\alpha}$ is defined by the following formula [3][11]:[12]

$$\hat{\alpha} = \arg\max_{\alpha} P(\boldsymbol{X}; \alpha) = \arg\max_{\alpha} \sum_{\boldsymbol{r}} P(\boldsymbol{X}|\boldsymbol{r}) P(\boldsymbol{r}; \alpha) \tag{11}$$

When the prior $P(\boldsymbol{r}; \alpha)$ is a conjugate prior for the likelihood $P(\boldsymbol{X}|\boldsymbol{r})$, $\hat{\alpha}$ can be computed analytically. Unfortunately, the likelihood function for R4S-EB is quite complicated. To handle this, I chose to use the expectation-maximization (EM) algorithm, initializing $\alpha^{(0)} = 1$ and iteratively computing:

$$\boldsymbol{r}^{(i)} = E[\boldsymbol{r}|\boldsymbol{X}; \alpha^{(i)}] \tag{12}$$

$$\alpha^{(i+1)} = \arg\max_{\alpha} P(\boldsymbol{r}^{(i)}; \alpha) \tag{13}$$

Equation 12, the E step of the algorithm, is computed via the R4S-EB authors' discretized Bayesian inference method described earlier. Equation 13, the M step, is computed as $\alpha = \frac{1}{Var(\boldsymbol{r})}$. These steps are repeated until the convergence of successive values for the marginal probability $P(\boldsymbol{X}, \boldsymbol{r})$, which

[9] One might want to compute $P(rt)$ for thousands of different r if MCMC methods of inference are used.

[10] Thus, I compute $P(rt)$ for only $K = 16$ different r, albeit for all branches in the given tree.

[11] *cf.* section 3.1.

[12] I omit the other hyperparameters τ, π, Q present in Equation 3. I also use $\boldsymbol{X}$ and $\boldsymbol{r}$ to represent the vectors $\{X_1, ..., X_L\}$ and $\{r_1, ..., r_L\}$, respectively.

is computed during each iteration as part of the E step.

In brief summary, R4S-EB can be implemented by specifying a routine to ingest π, Q by appropriately handling PAML format substitution model parameter files (Section 3.2.1), a routine to discretize the gamma prior on r to generate r_k for summing the posterior over (Section 3.2, end), a routine to compute the likelihood given r_k, π, Q, τ via postorder tree traversal ([28] section II) and the appropriate computation of the matrix exponential (Section 3.2.2), and a routine to perform EM over the inferred rates, optimizing the value of $r|\alpha$ (Section 3.2.3). I implement all of these steps and include them in my software package, discussed in Section 5.

4. Evaluation of Key Algorithms

In this section, I evaluate the prediction accuracy of INTREPID and R4S-EB by running their original algorithms, as well as variations to their original algorithms, against two externally labelled test datasets. Before I dive into the details of my tests, however, I first address the crucial question of which evaluation measures I chose to use, and why.

To start, I note that the best evaluation measures should, as directly as possible, measure the usefulness of a scorer to the interpreting biologist's intended task. The original motivation of scoring sites for sequence conservation was that sequence conservation scores could likely help biologists locate a protein's functionally important residues. Given scores produced by a sequence conservation scorer, then, a biologist would probably sort for the most highly conserved (and thus likely functional) sites, and then manually inspect them to assess their true functional importance.[13]

This use procedure implies that biologists should be mainly interested in (1) maximizing the fraction of functional sites they will see by going through all positive labels, (2) without having too many false positives appear in those positive labels.[14]

[13] A biologist might also use a scorer to inspect the inferred level of sequence conservation at predetermined sites. I deem the first use case as more common, however.

[14] In [2], the authors seem to have designed a test statistic to measure for (2) in their "rank analysis" method of evaluating scorers. That evaluation measure involves recording what fraction of the top 30 (or so) scored columns are functionally important. The authors do not design a test statistic to measure for (1), but intuitively, it seems like a desirable feature as well. Regardless, (1) and (2) turn out to be precisely precision and recall, respectively, and so it is actually quite common to want to optimize a classifier for these two features.

Many earlier evaluations of sequence conservation methods use ROC curves to assess a scorer's performance [2, 27]. It is not clear to me why ROC curves are particularly useful: Neither [2] nor [27] justify why ROC curves in particular were used, and I argue that the typical reasons one might choose to use ROC curves do not make sense in the context of using positive scores from a scorer to find true functionally important residues. First, one might choose to use an ROC curve to evaluate a classifier because ROC curves are independent of the prior distribution of positive and negative test labels [8]. For any real applications, however, scorers will be analyzing entire protein sequences, for which we have the fixed prior that ~ 1 to 10% of all sites will be actually functional (*cf.* Table 1). Second, ROC curves are generally looked at as a standard and fairly interpretable way of measuring the performance of classifiers against a binary test set. However, ROC curves plot the number of positive sites correctly identified (TPR; vertical axis) for different levels of error tolerance on non-functional sites (FPR; horizontal axis), implying that scorers that perform well on ROC curves are those that correctly label more actual positives for each error rate tolerance on non-functional sites. But features (1) and (2) only care about a scorer's error rate on non-functional sites insofar as that rate affects how often false positives are encountered during inspections of positively labeled sites. This is not an obvious relationship at all: The ROC evaluates the FPR $P(\hat{Y}=1|Y=0)$, but biologists only care about its effect on $P(Y=0|\hat{Y}=1)=\frac{P(\hat{Y}=1|Y=0)\,P(Y=0)}{P(\hat{Y}=1)}$. I thus suggest finding a more easily interpretable evaluation measure.

The precision-recall (PR) curve [7] seems perfectly designed for optimization of the features desired. The PR curve plots the precision $P(Y=1|\hat{Y}=1)$, or the fraction of labeled positives that are truly functional, against the recall $P(\hat{Y}=1|Y=1)$, or different fractions of functional sites that one desires to be labeled positive. The trade-off between precision and recall is hence exactly the same trade-off between (1) and (2). Conveniently, a biologist can intuitively use the PR curves of different scorers to discover which scorer is best for her purposes: *The biologist can set a desired minimum fraction of functional sites in a sequence that she wants flagged positive for review (x-axis), and then view the PR curves of different scorers to learn what fraction of the positive labels she is inspecting are actually expected to be functional sites (y-axis).* It should be obvious that the PR

	csa	**ec**
# alignments*	657	3071
Avg. # sequences per alignment*	35	39
Avg. # sites per alignment	353	350
Avg. % positive sites per alignment	1.1%	6.4%

Table 1: Summary of the catalytic site (CSA) and ligand-binding site (EC) labeled datasets from [2]. Note that very few sites in either dataset are flagged as functional, although over 6 times more are flagged in EC than in CSA. ∗I perform some filtering on the original datasets; see the Appendix for some details on my methods.

curve thus makes for an appropriate evaluation measure for sequence conservation scorers.

In Sections 4.1-4.3, I use these PR curves to evaluate the results of different variations of INTREPID and R4S-EB on externally labeled test data. As a brief note, the experimental data I use are Capra and Singh 2007's two datasets of (1) alignments labeled with catalytic sites ('CSA'), and (2) alignments labeled with ligand-binding sites ('EC') [2]. I choose these datasets because they were successfully used as ground truth in [2] to evaluate a number of basic heuristic sequence conservation methods.[15] Table 1 displays a basic summary of the datasets' contents. The reader should take note that only a small fraction of sites – 1.1% and 6.4% respectively – in CSA and EC are actually labeled as functional.

4.1. Comparison of 'vanilla' INTREPID and R4S-EB

I start by comparing INTREPID and R4S-EB's relative performance when they are run using their 'vanilla' published specifications. For each of the algorithms, I compute their unnormalized as well as per-sequence normalized scores on each of the datasets CSA and EC. I plot the scores' PR and ROC curves in Figure 3; for this test I draw ROC curves even though I do not find them to be useful because ROC curves are extensively relied upon in earlier evaluations in literature [2, 27], and I want to be able to visually compare my results with the published results.[16] Note that I refer to the normalized versions of INTREPID and R4S-EB as INTREPID-norm and R4S-EB-norm,

[15]In the same paper, Capra and Singh also provide a third dataset of protein-protein interfaces ('PPI'), but they find that none of the sequence conservation methods they analyzed could successfully predict PPI labels.

[16]*cf.* [27] fig. 2 vs. the bottom left figure here.

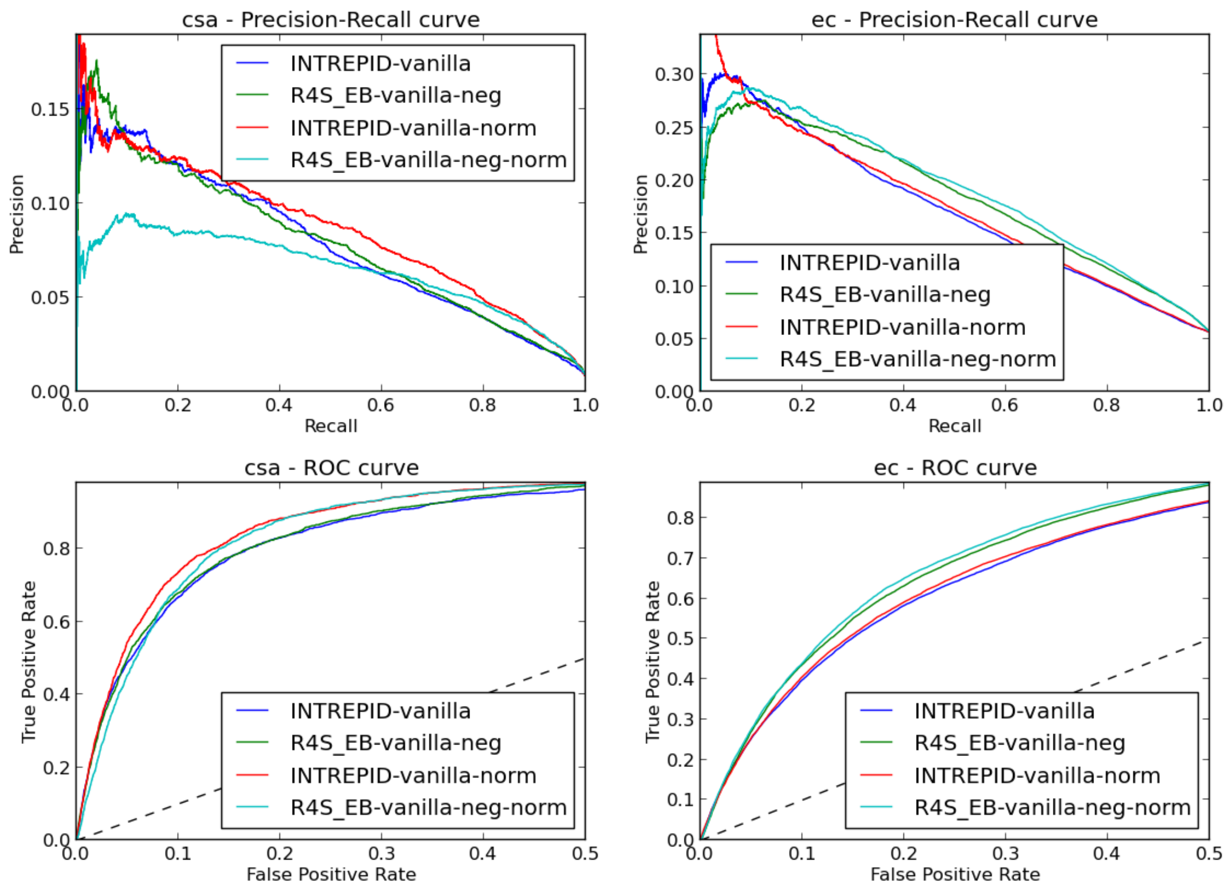


Figure 3: Precision-recall and ROC curves for the vanilla implementations of INTREPID and R4S-EB, with curves for both unnormalized and normalized scores shown.

respectively.

I suggest making the following observations:

1. *INTREPID is **NOT** "significantly more accurate than other sequence-based methods"*.[17] According to the CSA dataset's PR and ROC curves (top left), INTREPID-norm and INTREPID do both outperform R4S-EB-norm and R4S-EB, respectively. However, this outcome is flipped for the EC dataset (top right). Most naively, this might be due to INTREPID predicting catalytic residues more accurately and R4S-EB predicting ligand-binding sites more accurately; I suggest, however, that this discrepancy is more likely due to the way in which CSA and EC labels were assigned. Labels in CSA are assigned so that only amino acids directly involved in interactions with catalysts are flagged; labels in EC are assigned so that any amino acids within 4 Å of

[17]quote from [27], *cf.* section 3.1.2.

ligand atoms are flagged [2]. EC thus uses a somewhat looser definition of functionality than CSA.[18] This discrepancy could thus be accounted for if, for example, R4S-EB recognized sites immediately adjacent to functional sites but INTREPID did not.

Should my hypothesis be valid, this suggests that a more precise definition of site functionality (i.e., a choice of either the CSA or the EC definition) is needed if one were to want to make a definite determination as to which of INTREPID and R4S-EB's is more accurate. Additionally, the performance of both methods might possibly be improved by applying the appropriate neighbor smoothing or sharpening filter.

2. *Scores have better predictive performance when they are normalized per sequence.* With the very notable exception of R4S-EB-norm on CSA, all normalized versions of the algorithms outperform their unnormalized counterparts. This suggests that the distributions of scores over sites within different proteins can vary significantly, *and that proteins – or at least the enzymatic proteins in these datasets – should be treated as each having some fixed fraction of functional sites*.[19] As to the notably inferior performance of R4S-EB-norm on CSA but not EC, I cannot think of a plausible reason. I leave this puzzle open to bug checking and future work.

3. *The state-of-the-art implemented methods for rating sequence conservation produce scores that are 75-90% noise.*[20] Precision for both INTREPID and R4S-EB, overall, is very low. In general, depending on which of the CSA or EC definitions of site functionality one uses, only 10% to 25% of labeled positives are actually functional sites. This is still 10x and 5x better than a random guess for CSA or EC, respectively, but it also makes for positive predictions that contain up to 9 times as many non-functional sites as functional sites. It would be interesting to compare INTREPID and R4S-EB relative noisiness of other methods for inferring site functionality.

All of these observations come with the caveat that the statistical significance of the results has not been analyzed. It should be quite easy to implement statistical significance assessment using my

[18]This is likely the reason for EC's higher average per-sequence frequency of actual functional sites in Table 1.

[19]If we normalize the scores per sequence, the same fraction of sites from every sequence are labeled as positives no matter what positive-negative threshold is set. Since normalized scores give us better performance than absolute scores, we can infer that this model is more accurate than the alternative (i.e. that different sequences have different numbers of absolute sites and absolute scores can help us distinguish such sequences).

[20]I define noise to be the fraction of positively labeled sites that are not true positives.

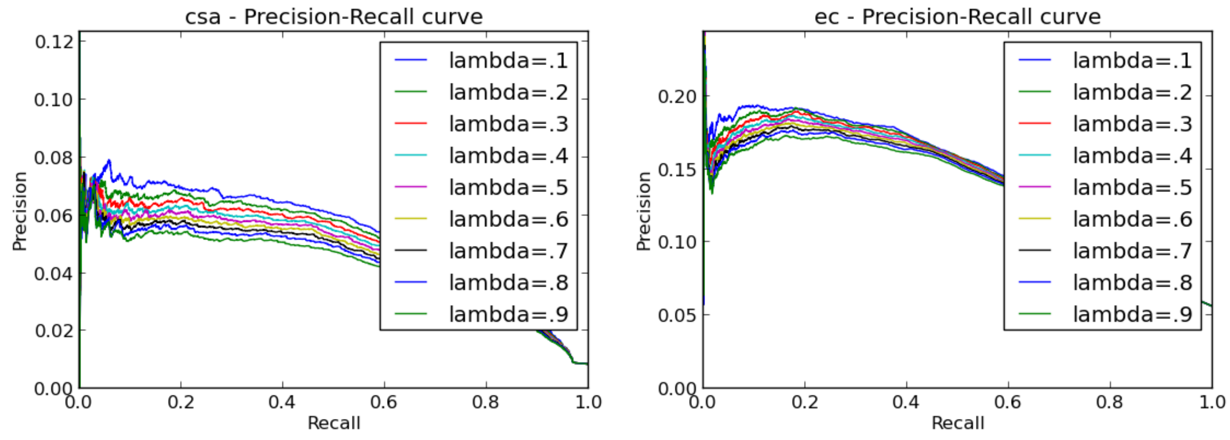


Figure 4: Precision-recall curves for different settings of the prior weight λ into the standalone Jensen-Shannon divergence method. As shown, setting $\lambda = 0.1$, the lowest value tested, gives the best performance.

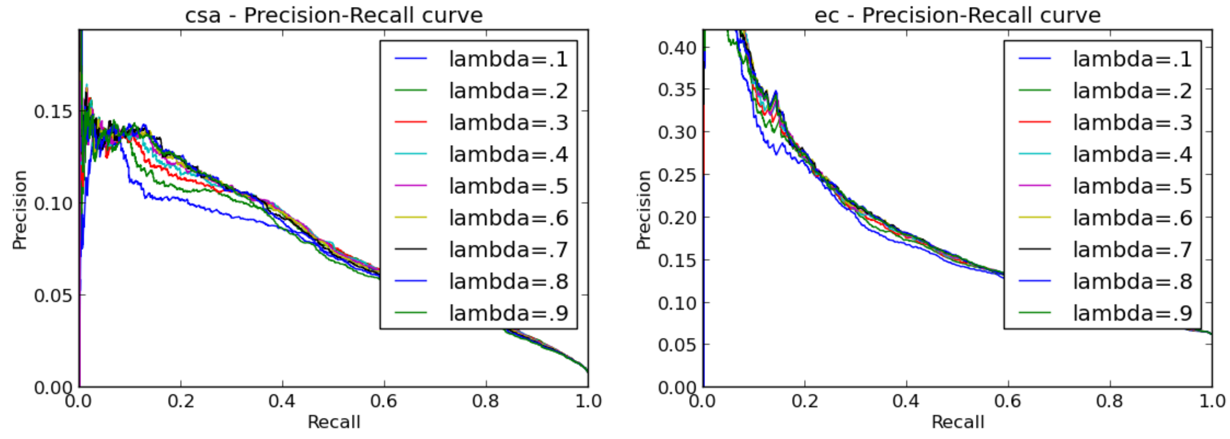


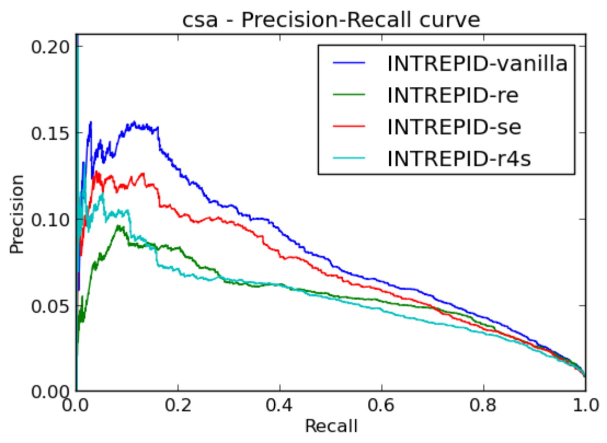
Figure 5: Precision-recall curves for different settings of the prior weight λ into the INTREPID algorithm. Unlike in Figure 4, changing λ does not seem to improve INTREPID's performance in any predictable way.

software package (Section 5), but I leave such implementation to future work.

4.2. Potential improvements to INTREPID

INTREPID takes in two easily substitutable inputs: (1) The λ prior weight in the Jensen-Shannon divergence (JSD) formula it uses to compute divergences over subtrees, and (2) the actual choice of the divergence measure it uses to compute divergences over subtrees itself. I examine alternate settings of each of these in turn.

Testing different λ: To motivate this test, I first refer to one of my observations, shown in Figure 4,



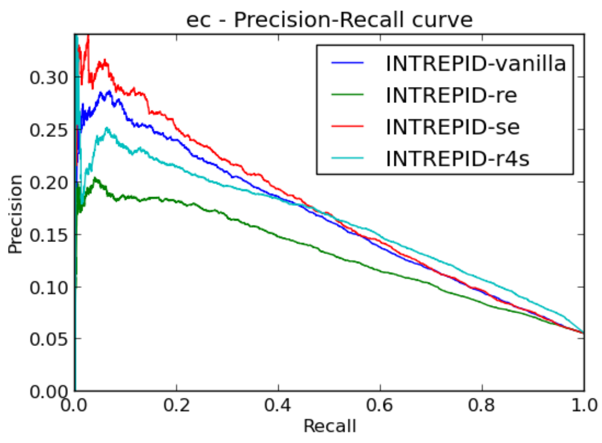


Figure 6: Precision-recall curves for different choices of the divergence measure in the INTREPID algorithm. Here, 're' refers to relative entropy, 'se' refers to Shannon entropy, and 'r4s' refers the negated output from R4S-EB. Shannon entropy interestingly outperforms the more complex JSD measure on the EC dataset.

that *the precision-recall performance of the standalone JSD method*[21] *improves with smaller settings of λ.* My observation conflicts with published findings – in [2], Capra and Singh recommend $\lambda = .5$ as giving their algorithm the best performance[22] – but since Capra and Singh do not make available the data supporting their recommendation, I stand by my discovery. As a side note, *I recommend setting $\lambda = 0.1$ when scoring for sequence conservation using the standalone JSD.* Interestingly, this result is actually supported by one theoretical interpretation of the JSD (see Appendix).

Unfortunately, inputting alternative λ into INTREPID does not improve INTREPID's performance. As shown in Figure 5, reducing the value of λ seems to have an unpredictable effect, or even a deleterious effect, on the performance of the algorithm. I simply display my data here, leaving discussion of possible reasons to future work.

Testing different divergence measures: This test was actually performed by the original authors of INTREPID. Here, I replicate that experiment (detailed in the supplementary material to their paper), testing different substitutions for the JSD as INTREPID's global divergence measure. In particular, I try substituting in the relative entropy measure, the Shannon entropy measure, and the negated

[21] where the score used is the JSD divergence over the root tree. See [2] for the paper originally introducing this scoring method.

[22] Comparing my results with Capra and Singh's results head-on may not seem entirely fair, since I use PR curves while they use ROC curves for evaluation. However, my results also hold using ROC curves; see the Appendix for figures.

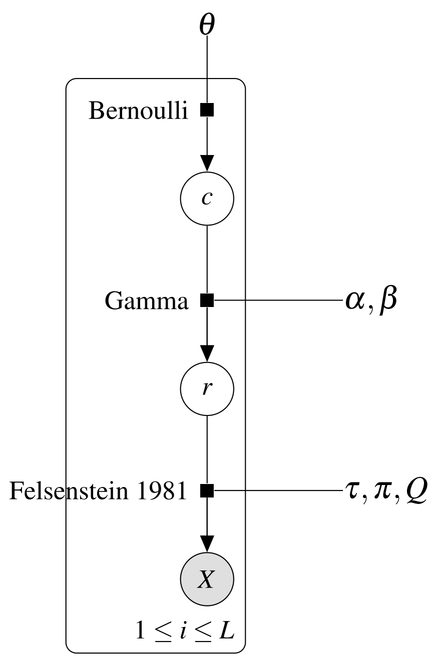


Figure 7: An alternate probabilistic model for R4S-EB. Each rate r, instead of being selected from a fixed prior, is selected from a hidden binary variable c describing the binary outcome of whether or not the site is functional. See Figure 1 for definitions of the other variables and distributions in this model.

output from R4S-EB. As shown in Figure 6, on the CSA dataset, JSD still gives INTREPID the best performance. On the EC dataset, however, Shannon entropy improves INTREPID's performance. I leave the exploring of whether this improved performance is significant, whether there is an interpretable cause, and whether this improved performance can be replicated on other datasets in a consistent and meaningful way, to future work.

4.3. Potential improvements to R4S-EB

Given the complexity of the R4S-EB algorithm, it is harder to test significant changes to the R4S-EB model. In place of performing multiple tests as I did for INTREPID, I critique several components of the R4S-EB probabilistic model here to suggest directions for future work. Points 1, 2, and 3 comment on the difficulty of altering certain parts of the model, but point 4 suggests a potential improvement that should not be overly difficult to implement and test.

1. *Substituting out the likelihood function:* The likelihood function used by R4S-EB is almost exactly Felsenstein's 1981 likelihood model for phylogenetic trees [9]. Even though calculating this likelihood is one of the biggest computational bottlenecks in R4S-EB (thus helping make it far slower than any of the heuristc conservation scorers [27]), it seems unlikely that this can be switched out considering the broad acceptance of this model in all sorts of applications of

phylogenetic trees [31, 32, 33, 24, 28].

2. *Substituting out the gamma prior:* As discussed in Section 3.2, the choice of the gamma distribution for the prior is based on a significant amount of literature validating the gamma's excellent fit on experimental data [29, 21, 31]. Mathematically, the gamma distribution also has sensible properties that make it work well as r's prior: It is skew left and has a long tail, representing the intuition that rapid evolution at a site can happen but is a very rare event; it has an adjustable variance, making it more flexible than another model with a long tail, the Poisson; and it selects from the range $[0, \infty)$, making it sensible to use to generate multiplicative rates. Given all these convenient properties, it seems difficult and unnecessary to switch out this distribution for anything else.
3. *Weakening the assumption that sites are mutually independent:* The model for R4S-EB's prior specifies that the rates for different sites are mutually independent. This assumption neglects window effects: For example, an amino acid's neighbors in its sequence might be involved in carrying out function with it together. While introducing inter-column dependencies as part the probabilistic model would likely make inference of the posterior too computationally demanding, it would be interesting to explore if heuristically windowing the inferred rates from this model (as a post-processing technique) would improve prediction performance.
4. *Introducing an additional hidden variable describing functionality/non-functionality:* It is interesting to note that while rates inferred from this model range in $[0, 20]$, one of the main motivations of the sequence conservation problem was to infer the binary outcomes of whether or not each site is functional. It might make sense to encode this as part of the probabilistic model used in R4S-EB. Such would involve introducing a hidden variable c specifying the functionality or non-functionality of a site; r would then be drawn from a gamma distribution on α_c, β_c where the choice of these parameters depends on the identity of the site's c.

5. ConsEval: Software enabling simple model implementation and evaluation

In Section 3, I detailed the models and implementation behind the INTREPID and R4S-EB algorithms. In Section 4, I shared results from a number of tests that I performed on aggregations of different runs of these algorithms. These two sections represent the two stages of model specification and model evaluation, and they are perhaps the most interesting steps of performing data analysis.

As most researchers are well aware, however, most time tends to get spent on the steps *between* these two stages. In particular, to get from model specification to model evaluation, one typically must:

1. Organize, parse, load, and otherwise pre-process input files and test files
2. Implement the models of interest, and (as is often desired) enable the adjustment of parameters as well
3. Write a routine to execute the implementations, ideally efficiently and in parallel, with output saved and organized in a sensible format
4. Write methods to evaluate and plot evaluations of that output

I implement Python package to simplify these tedious steps between specifying and evaluating protein functional site scorers. I call this package 'ConsEval' because it helps researchers more efficiently *Eval*uate sequence *Cons*ervation scoring methods. ConsEval abstracts away the repetitive parts of the above steps, specifically making it easier to:

1. Specify input datasets and parsing routines.
2. Implement any protein site scoring method, with adjustable parameters if desired.
3. Score a single alignment using one scoring method, or batch score (in parallel) a dataset of alignments using multiple scoring methods.
4. Implement and run evaluations of the models.

In other words, ConsEval provides a basic API for performing each of these steps. I outline how this API treats each of the four enumerated steps below.

5.1. Specifying input datasets and parsing routines

An input dataset normally consists of a directory of alignments and a directory of test labels. In ConsEval, datasets are specified by adding the following information, paired with a `dataset_name`, to `conseval.datasets.DATASET_CONFIGS`:

`aln_dir`	Directory containing alignment files, represented in standard CLUSTAL or FASTA format, with path relative to `./input/`
`test_dir`	Directory containing test labels, represented in whatever format desired, with path relative to `./input/`
`align_to_test_fn`	Function that takes in an alignment file's filename, and returns its corresponding test file's filename
`parse_testset_fn`	Function that takes in a test file's filename and the protein sequence as a list of amino acids, and returns its corresponding test labels

5.2. Implementing protein site scoring methods, with adjustable parameters

Protein scorers can be defining a new class `MyScorer` that subclasses from `conseval.scorer.Scorer`, and placing it in the module `scorers.my_scorer`. The folder `scorers` contains a number of implemented `Scorer`s, each of which serve as good examples for how to implement a `Scorer`.

In brief, `Scorer`s override the method `MyScorer._score(alignment)`, where `alignment` is an `conseval.alignment.Alignment` object to be scored and the return value should be a list of floats representing the scores of each site in the alignment. The most important fields of the `Alignment` object are as follows:

`msa`	List of sequences in the alignment, where sequences are lists of amino acid characters
`names`	Names of sequences in the alignment
`align_file`	File that the alignment was read from
`testset`	List of labels in the test set for the alignment, if the alignment was initialized with test file source

To define parameters, scorers override the class variable `MyScorer.params` with a list of `conseval.params.ParamDef`s. These `ParamDef`s can be initialized with a simple name of the parameter, default value for the parameter, function to parse the value of the parameter from a string (to enable setting parameters from the command line), an optional function to verify that the value of the parameter is valid, and an optional help message describing the parameter.[23]. Conveniently, parameters `param_name` in `MyScorer.params` can be set when the `MyScorer` object is initialized: `ms = MyScorer(param_name='param value')` automatically sets `ms.param_name` to be `'param value'`.

5.3. Executing scorers on single files or in batch mode

With just a `MyScorer` defined, the script `score.py` can be run as:

```
  score.py my_scorer aln_file -p param_name=param_value -p param2=0.1
```

This runs `MyScorer._score` and prints human-readable output of the scores for the alignment. As for batch scoring alignments, the script `batchscore.py` can be run as:

```
  batchscore.py yaml_config
```

`yaml_config` is a simple YAML configuration file listing the scorers that are to be run each listing their name (e.g., `my_scorer`), desired identifier (e.g., `my_scorer_today`), and params; and the datasets the scorers are to be run for.[24]. This runs the scorers in parallel with error handling, and outputs the scores for each alignment and for each scorer to `./output/batchscore-dataset_name/scorer_id/al`
These files can be easily read by `evaluate.py` script, briefed over in the next section.

5.4. Implementing and executing evaluators of output from batch mode scoring

Any evaluation routine can leverage the ConsEval framework by being defined as a function that takes in a `dataset_name` and any number of `scorer_id`s, written in a new module in `evaluators/`, where the module name must equal the function name. Suppose I wrote a function `evaluate_me`, which I save at `evaluators/evaluate_me.py`. Then, running the following command on the command line

```
  evaluate.py evaluate_me csa scorer_id1 scorer_id2
```

causes `evaluate_me('csa', 'scorer_id1', 'scorer_id2')` to be called.

[23] `ParamDef`s can also take in another input that allow them to load files. See the class's documentation for more details.
[24] An example YAML config is provided at `examples/example.yaml`

A function `get_batchscores(dataset_name, scorer_ids)` that takes in a dataset name and list of scorer names, and returns an iterator on alignments (which contain the test data as the `testset` field) and their corresponding scores, has been defined in `evaluate.py`. Most evaluator functions will need to call this `get_batchscores` function.

As can be seen, ConsEval helps to abstract away the most tedious steps in analyzing data for scoring protein sites for their sequence conservation. Both computational biology researchers interested in testing and evaluating different sequence conservation models, and lab biologists who might simply want to run `score.py` on a single alignment, can benefit from this package.

ConsEval can be accessed as a Github project on the Web at `https://github.com/jwayne/iw-conservation`.[25]

6. Conclusion

The accurate identification of the function of arbitrary proteins is an enormous outstanding problem in biology. Sequence conservation methods can be used to help tackle this problem by helping to identify the functional sites of any protein given its sequence; these functional sites can in turn be used as strong signals of the protein's function. In this paper, I helped develop a better understanding of these sequence conservation methods by methodically detailing the INTREPID and Rate4Site empirical Bayes algorithms, carefully evaluating several key aspects of these algorithms, and developing a Python package, ConsEval, that can be used to now more easily continue the kinds of evaluations I performed. Several key results of my evaluation are my finding that INTREPID does not truly outperform other sequence conservation scoring methods, my discovery of an improved parameter setting for the Jensen-Shannon conservation scoring method, my careful justification of precision-recall curves for evaluating the performance of sequence conservation scorers, and my suggestion of a number of questions to explore in future work. It is hoped, in particular, that ConsEval will prove useful in future research on functional site identification methods.

[25] As a side note, ConsEval uses a very minor amount of code from Capra and Singh's code supplementing their paper [2]. This code exists mostly in the implementations of Capra and Singh's information theoretic scorers in `scorers.cs07`.

References

[1] H. Ashkenazy *et al.*, "ConSurf 2010: calculating evolutionary conservation in sequence and structure of proteins and nucleic acids," *Nucleic Acids Res.*, vol. 38, no. Web Server issue, pp. W529–533, Jul 2010.

[2] J. A. Capra and M. Singh, "Predicting functionally important residues from sequence conservation," *Bioinformatics*, vol. 23, no. 15, pp. 1875–1882, Aug 2007.

[3] B. Carlin and T. Louis, "Bayes and empirical bayes methods for data analysis," *Statistics and Computing*, vol. 7, no. 2, pp. 153–154, 1997. Available: http://dx.doi.org/10.1023/A%3A1018577817064

[4] G. Casari, C. Sander, and A. Valencia, "A method to predict functional residues in proteins," *Nat. Struct. Biol.*, vol. 2, no. 2, pp. 171–178, Feb 1995.

[5] G. Celniker *et al.*, "Consurf: Using evolutionary data to raise testable hypotheses about protein function," *Israel Journal of Chemistry*, vol. 53, no. 3-4, pp. 199–206, 2013. Available: http://dx.doi.org/10.1002/ijch.201200096

[6] S. Chakrabarti and C. J. Lanczycki, "Analysis and prediction of functionally important sites in proteins," *Protein Science*, vol. 16, no. 1, pp. 4–13, 2007. Available: http://dx.doi.org/10.1110/ps.062506407

[7] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *In ICML '06: Proceedings of the 23rd international conference on Machine learning*. ACM Press, 2006, pp. 233–240.

[8] T. Fawcett, "An introduction to roc analysis," *Pattern Recogn. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006. Available: http://dx.doi.org/10.1016/j.patrec.2005.10.010

[9] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach," *J. Mol. Evol.*, vol. 17, no. 6, pp. 368–376, 1981.

[10] A. Garrido, "About some properties of the kullback-leibler divergence," *Advanced Modeling amiscnd Optimization*, vol. 11, no. 4, pp. 571–578, 2009. Available: http://camo.ici.ro/journal/vol11/v11d15.pdf

[11] F. Glaser *et al.*, "A method for localizing ligand binding pockets in protein structures," *Proteins*, vol. 62, no. 2, pp. 479–488, Feb 2006.

[12] A. Gutteridge, G. J. Bartlett, and J. M. Thornton, "Using a neural network and spatial clustering to predict the location of active sites in enzymes," *J. Mol. Biol.*, vol. 330, no. 4, pp. 719–734, Jul 2003.

[13] T. Hawkins and D. Kihara, "Function prediction of uncharacterized proteins," *J Bioinform Comput Biol*, vol. 5, no. 1, pp. 1–30, Feb 2007.

[14] H. Kishino, T. Miyata, and M. Hasegawa, "Maximum likelihood inference of protein phylogeny and the origin of chloroplasts," *Journal of Molecular Evolution*, vol. 31, no. 2, pp. 151–160, 1990. Available: http://dx.doi.org/10.1007/BF02109483

[15] C. Kosiol and N. Goldman, "Different versions of the Dayhoff rate matrix," *Mol. Biol. Evol.*, vol. 22, no. 2, pp. 193–199, Feb 2005.

[16] S. Q. Le and O. Gascuel, "An improved general amino acid replacement matrix," *Molecular Biology and Evolution*, vol. 25, no. 7, pp. 1307–1320, 2008. Available: http://mbe.oxfordjournals.org/content/25/7/1307.abstract

[17] O. Lichtarge, H. R. Bourne, and F. E. Cohen, "An evolutionary trace method defines binding surfaces common to protein families," *Journal of Molecular Biology*, vol. 257, no. 2, pp. 342 – 358, 1996. Available: http://www.sciencedirect.com/science/article/pii/S0022283696901679

[18] I. Mayrose *et al.*, "Comparison of site-specific rate-inference methods for protein sequences: empirical Bayesian methods are superior," *Mol. Biol. Evol.*, vol. 21, no. 9, pp. 1781–1791, Sep 2004.

[19] C. Moler and C. V. Loan, "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later," *SIAM Review*, vol. 45, no. 1, pp. 3–49, 2003. Available: http://link.aip.org/link/?SIR/45/3/1

[20] S. D. Mooney *et al.*, "Structural characterization of proteins using residue environments," *Proteins*, vol. 61, no. 4, pp. 741–747, Dec 2005.

[21] M. Nei and T. Gojobori, "Simple methods for estimating the numbers of synonymous and nonsynonymous nucleotide substitutions," *Mol. Biol. Evol.*, vol. 3, no. 5, pp. 418–426, Sep 1986.

[22] C. Ouzounis *et al.*, "Are binding residues conserved?" *Pac Symp Biocomput*, pp. 401–412, 1998.

[23] Philip Resnik and Eric Hardisty, "Gibbs Sampling for the Uninitiated," University of Maryland, College Park, Tech. Rep. LAMP-TR-153, 2010.

[24] T. Pupko *et al.*, "Rate4Site: an algorithmic tool for the identification of functional regions in proteins by surface mapping of evolutionary determinants within their homologues," *Bioinformatics*, vol. 18 Suppl 1, pp. S71–77, 2002.

[25] P. Radivojac *et al.*, "A large-scale evaluation of computational protein function prediction," *Nat. Methods*, vol. 10, no. 3, pp. 221–227, Mar 2013.

[26] S. Sankararaman, B. Kolaczkowski, and K. Sjolander, "INTREPID: a web server for prediction of functionally important residues by evolutionary analysis," *Nucleic Acids Res.*, vol. 37, no. Web Server issue, pp. W390–395, Jul 2009.

[27] S. Sankararaman and K. Sjolander, "INTREPID–INformation-theoretic TREe traversal for Protein functional site IDentification," *Bioinformatics*, vol. 24, no. 21, pp. 2445–2452, Nov 2008.

[28] H. Schabauer *et al.*, “Slimcodeml: An optimized version of codeml for the branch-site model,” in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, 2012, pp. 706–714.

[29] T. Uzzell and K. W. Corbin, “Fitting discrete probability distributions to evolutionary events,” *Science*, vol. 172, no. 3988, pp. 1089–1096, Jun 1971.

[30] E. Weisstein, “"incomplete gamma function." From MathWorld—A Wolfram Web Resource.” Available: http://mathworld.wolfram.com/IncompleteGammaFunction.html

[31] Z. Yang, “Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites,” *Mol. Biol. Evol.*, vol. 10, no. 6, pp. 1396–1401, Nov 1993.

[32] Z. Yang, “A space-time process model for the evolution of DNA sequences,” *Genetics*, vol. 139, no. 2, pp. 993–1005, Feb 1995.

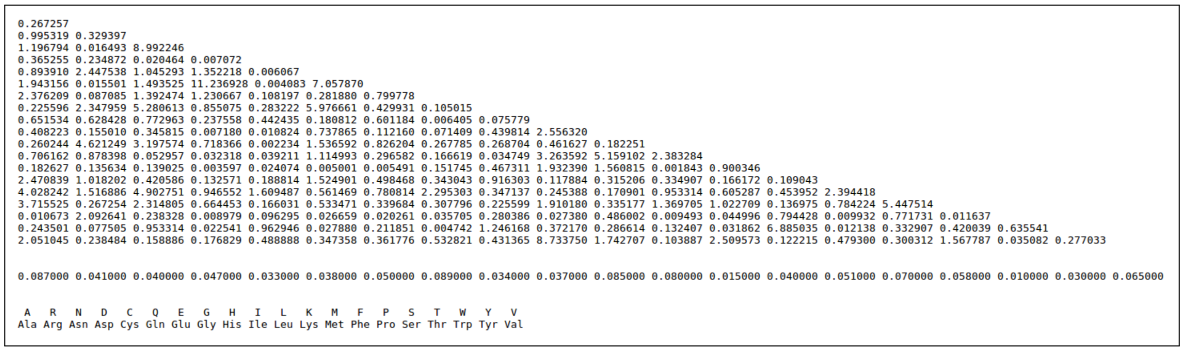
[33] Z. Yang, “Paml 4: Phylogenetic analysis by maximum likelihood,” *Molecular Biology and Evolution*, vol. 24, no. 8, pp. 1586–1591, 2007. Available: http://mbe.oxfordjournals.org/content/24/8/1586.abstract

[34] H. Yao *et al.*, “An accurate, sensitive, and scalable method to identify functional sites in protein structures,” *Journal of Molecular Biology*, vol. 326, no. 1, pp. 255 – 261, 2003. Available: http://www.sciencedirect.com/science/article/pii/S0022283602013360

[35] E. Youn *et al.*, “Evaluation of features for catalytic residue prediction in novel folds,” *Protein Science*, vol. 16, no. 2, pp. 216–226, 2007. Available: http://dx.doi.org/10.1110/ps.062523907

7. Appendix

7.1. Example of substitution model file in PAML format



```
0.267257
0.995319 0.329397
1.196794 0.016493 8.992246
0.365255 0.234872 0.020464 0.007072
0.893910 2.447538 1.045293 1.352218 0.006067
1.943156 0.015501 1.493525 11.236928 0.004083 7.057870
2.376209 0.087085 1.392474 1.230667 0.108197 0.281880 0.799778
0.225596 2.347959 5.280613 0.855075 0.283222 5.976661 0.429931 0.105015
0.651534 0.628428 0.772963 0.237558 0.442435 0.180812 0.601184 0.006405 0.075779
0.408223 0.155010 0.345815 0.007180 0.010824 0.737865 0.112160 0.071409 0.439814 2.556320
0.260244 4.621249 3.197574 0.718366 0.002234 1.536592 0.826204 0.267785 0.268704 0.461627 0.182251
0.706162 0.878398 0.052957 0.032318 0.039211 1.114993 0.296582 0.166619 0.034749 3.263592 5.159102 2.383284
0.182627 0.135634 0.139025 0.003597 0.024074 0.005001 0.005491 0.151745 0.467311 1.932390 1.560815 0.001843 0.900346
2.470839 1.018202 0.420586 0.132571 0.188814 1.524901 0.498468 0.343043 0.916303 0.117884 0.315206 0.334907 0.166172 0.109043
4.028242 1.516886 4.902751 0.946552 1.609487 0.561469 0.780814 2.295303 0.347137 0.245388 0.170901 0.953314 0.605287 0.453952 2.394418
3.715525 0.267254 2.314805 0.664453 0.166031 0.533471 0.339684 0.307796 0.225599 1.910180 0.335177 1.369705 1.022709 0.136975 0.784224 5.447514
0.010673 2.092641 0.238328 0.008979 0.096295 0.026659 0.020261 0.035705 0.280386 0.027380 0.486002 0.009493 0.044996 0.794428 0.009932 0.771731 0.011637
0.243501 0.077505 0.953314 0.022541 0.962946 0.027880 0.211851 0.004742 1.246168 0.372170 0.286614 0.132407 0.031862 6.885035 0.012138 0.332907 0.420039 0.635541
2.051045 0.238484 0.158886 0.176829 0.488888 0.347358 0.361776 0.532821 0.431365 8.733750 1.742707 0.103887 2.509573 0.122215 0.479300 0.300312 1.567787 0.035082 0.277033


0.087000 0.041000 0.040000 0.047000 0.033000 0.038000 0.050000 0.089000 0.034000 0.037000 0.085000 0.080000 0.015000 0.040000 0.051000 0.070000 0.058000 0.010000 0.030000 0.065000


 A   R   N   D   C   Q   E   G   H   I   L   K   M   F   P   S   T   W   Y   V
Ala Arg Asn Asp Cys Gln Glu Gly His Ile Leu Lys Met Phe Pro Ser Thr Trp Tyr Val
```

Figure 8: Example of an amino acid substitution model provided in PAML format.

Figure 8 shows a snapshot of an amino acid substitution model provided in PAML format. The first 19 lines contain the lower triangle of the symmetric exchangeability parameters S without the diagonals, and the next non-empty line contains the amino acid frequency distribution π. Many such files for popular substitution models can be found at http://www.ebi.ac.uk/goldman/dayhoff/.

7.2. Adjustments made to the CSA and EC datasets

Below, I list the major adjustments I make to the datasets 'CSA' and 'EC' available at [2]. For further details, I refer the reader to my code, which is available online with the package overviewed in Section 5.

- Alignments that have more than half of their columns consist of more than half gaps are thrown out. This weak sanity check filters out 3/660 and 1/3072 alignments from CSA and EC, respectively.
- I limit the number of sequences per alignment to 50 to reduce the time needed to prepare phylogenetic trees for INTREPID and R4S-EB. I perform this filtering by removing the first sequence of each alignment as the protein of interest, randomly sampling up to 49 different sequences from the remaining sequences in the alignment, and combining the result back with the protein of interest.

- CSA test labels correctly align one-to-one with columns in their corresponding alignments. However, EC test labels sometimes contain gaps and insertions relative to their corresponding alignments, and therefore do not. I perform a heuristic best-effort matching of EC test labels to their corresponding EC alignments – see my code for the detailed method. This results in approximately a dozen failed matches of labels with alignments, which I throw out from the 3071-alignment dataset.

7.3. Tests on different λ settings for the standalone JSD method

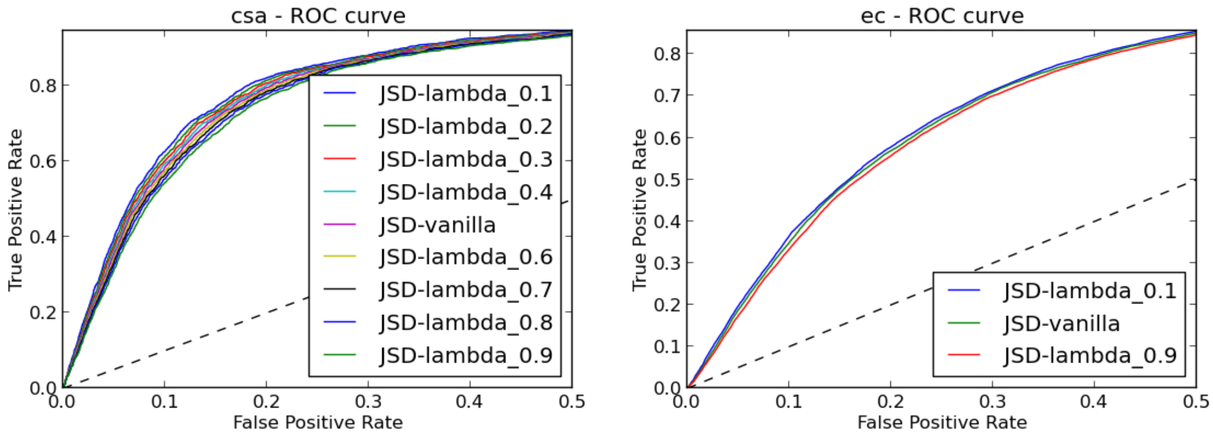


Figure 9: ROC curves for different settings of the prior weight λ on the standalone JSD algorithm. As shown and as is consistent with the PR curves in Figure 4, setting $\lambda = 0.1$, the lowest value tested, gives the best prediction performance.

Figure 9 shows ROC curves for my tests of different settings of λ for the standalone version of the Jensen-Shannon divergence method. Recall that PR curves for this same test are plotted in Figure 4.

As for the theoretical intuition that smaller values of than $\lambda = 0.5$ should give better results for scoring sequence conservation, recall the following:

- The JSD between two distributions p, q with prior weight λ represents the information gained upon learning the correct distribution of any sample if one's prior on the two distributions was $(\lambda, 1-\lambda)$ [10].
- The JSD we compute uses p equal to the distribution of amino acids in the column of interest,

and $q = \pi$ equal to the background distribution of amino acids.

- The JSD scores conserved columns highly because p is very dissimilar from π for those columns.

Hence, one might obtain better results with the JSD when λ to closer to our prior that an amino acid is conserved. As summarized for our data in Table 1, this prior is on the order of 1% to 6%.