Exploiting Large Datasets Improves Accuracy Estimation for Multiple Sequence Alignment

Luis Cedillo, Hector Richart Ruiz, Dan DeBlasio
Computer Science Department, University of Texas at El Paso

Abstract

Multiple sequence alignment is a fundamental problem in bioinformatics, which is why a large number of tools are used to align sequences under a prescribed (biologically inspired) objective function. Often practitioners use software's default parameters to align sequences. However, a different parameter setting may provide a much higher-quality alignment for the specific set of input sequences. One highly-accurate method of choosing parameter vectors for specific input is Parameter Advising, which selects from a set of alignments produced using a carefully constructed collection of parameter configurations. To choose among the candidate alignments, it would be ideal to use each alignment's accuracy, but in practice, a reference from which to calculate this measure is not available. One must *estimate* the accuracy of different alignments to rank them. The accuracy estimator `Facet` (short for Feature-based accuracy estimator) computes a single estimate of accuracy as a linear combination of efficiently-computable feature functions. We introduce `Facet-NN` and `Facet-LR` which both use the same underlying feature functions as `Facet` (as they were shown to be accurate), but since they are built on top of highly efficient machine learning protocols, they can take advantage of a much larger training corpus. Not only does this evolution allow us to train on much larger datasets, it produces an estimator that is more correlated with true accuracy. When used in Parameter Advising, `Facet-NN` and `Facet-LR` show an increase of 6% over using only the default parameter vector, which is a 2% increase over using `Facet` for the same task.

1 Introduction

The alignment of multiple biological, specifically protein, sequences is a key step in much of the analysis done by computational biologists. These alignments provide key insights into the correlation of various regions in the protein, and can be a guide to determining evolutionary history. From a computational standpoint though the problem is complicated, and in fact, it was proven that finding an optimal alignment of a set of sequences is NP-Complete [35, 19]. Because of this juxtaposition of the importance and complexity of the problem there are many existing heuristics that find *good* (though not optimal) multiple sequence alignments (multiple forms of `Clustal` [31, 30], `MAFFT` [16], `MUSCLE` [13], `T-Coffee` [25], `Opal` [36] among others)

None of these tools address a fundamental issue: the multiple sequence alignment problem requires as part of its input a user-defined objective function. While there are several well-studied objectives, the most common of which is sum-of-pairs, this still leaves the open question of the function's parameters: the rewards and penalties associated with each operation used to construct the alignment. Thus when new users, or even experienced users with a new class of inputs, approach the task of creating a multiple sequence alignment, they are required to choose a value for each of the tunable parameters of the tool (the collection of parameter values used is called a *parameter vector*). Making the *wrong* choice can have a highly detrimental impact on downstream analysis; manually tuning these algorithm values in a systematic manner requires not only a more-than-cursory understanding of the underlying application and the data, but also substantial amounts of time. Because of this most users rely upon the *default* parameter vector included with each tool. These defaults are specifically set to provide good results on average across all types of input, but the most interesting biological datasets are typically far from average.

Most alignment objective functions, and in turn the tools that utilize them, have parameters for both the penalty assessed for adding gaps into a sequence and the penalty/reward for the substitution (match/mismatch) of any two amino acids (residue). This means a user needs to specify For the 20-character amino acid alphabet used for protein sequences, 190 individual values need to be specified for the substitutions alone. Thankfully this is a well-studied problem, and there are many highly-accurate pre-calculated databases of substitution scores, called *substitution matrices*. Typically users are given a choice from a class of matrices (`VTML` [24], `BLOSUM` [15], and `PAM` [8]). Within each class, several matrices are available optimized for sequences with a specific level of entropy. When substitution matrices are used, the total number of tunable parameters is greatly reduced, simplifying (though not eliminating) the parameter choice problem.

One confounding factor for making an informed parameter vector choice for multiple sequence alignment is that unlike some domains, such as transcript assembly which has a readily available quality measurement which can be used on any input [11], the standard method for assessing accuracy requires comparison to a reference (ground-truth) alignment. Thus, the accuracy of a computed alignment can only be measured on so-called *benchmark* sequence sets. Typically these benchmarks are constructed by aligning the three-dimensional structure of proteins to find amino acids that are highly correlated. Those groups of amino acids (one from each protein) that are within some threshold of distance in the 3D alignment are turned into complete columns of the sequence alignment. These anchoring positions are called core columns. Accuracy of a computed alignment is then calculated as the fraction of substitutions from the core columns of the reference alignment that are recovered. Several databases of these benchmarks exist and are regularly used to compare multiple sequence alignment tools [4, 14, 29, 34, 3].

When a new (non-benchmark) sequence set is presented and aligned, there is no reference alignment from which to measure accuracy. In this case, one is left to estimate the accuracy of the computed alignment. Several tools exist to do this estimation they can be classified into two major classes: *scoring-function-based* tools [17, 5, 32, 2] which calculate a value based on the combination of measurable attributes of the alignment alone; and *support-based* tools [22, 21, 28] which use a collection of alignments over the same sequences to label each one in the set.

Using one of the tools above to estimate accuracy, a user can then begin to search for the optimal parameter vector for their input. While this can be done by hand or using an off-the-shelf iterative optimization mechanism (such as coordinate ascent [37]), this is typically still quite time consuming. To automate this process DeBlasio and Kececioglu [10] developed the *Parameter Advising* framework, which without additional wall-clock time if the appropriate resources are available can make input-specific parameter choices for multiple sequence alignment (details of an advisor are contained in Section 2 to make this work fully self-contained). In that work, they show that the advisor using the `Opal` alignment tool and the `Facet` accuracy estimator proved to have the highest advising accuracy.

In this study, we present `Facet-NN` and `Facet-LR`; two new scoring-function-based accuracy estimators which reimagine the original `Facet` estimator by using modern machine learning techniques for optimization, rather than combinatorial optimization, to exploit the much larger datasets we have produced. Although `Facet-NN` and `Facet-LR` rely on the same underlying set of efficiently-computable feature functions as `Facet`, their parameter advising performance is substantially better. They show an average increase in advising accuracy of 6% over using only the default parameter choice, this is more than a 2% increase over original `Facet` on estimator-aware advisor sets. The newly developed estimators also show a much stronger correlation with true accuracy due to the fact that we have been able to optimize for this task specifically.

2 Summary of Prior Work

Parameter Advising, depicted in Figure 1, takes the same input as the underlying scientific application and returns a single result. So to the end-user it appears to be just another tool to solve an existing problem but in this case one without any tunable parameters. But abstracted from this user is a process that automatically chooses a parameter vector for the specific input that provides a higher-than-default accuracy.

An advisor contains two key components: a set of candidate parameter vectors, called an *advisor set*; and an accuracy estimation tool used to choose from among those vectors, called an *advisor estimator*.

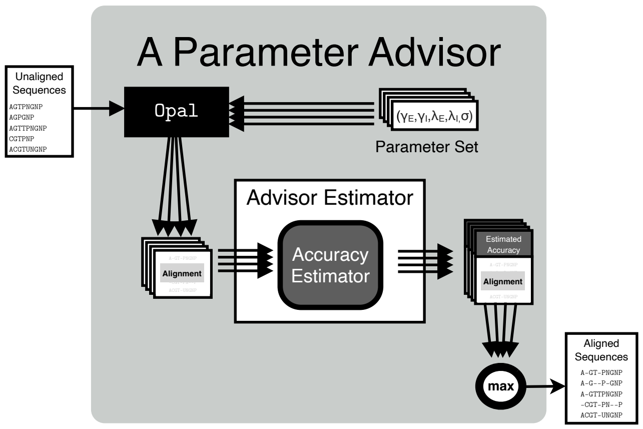

Figure 1: The parameter advising process.

The actual construction of these two components is described in detail below.

The Parameter Advising procedure is as follows:

- The input is combined with each of the parameter vectors in the advisor set (and the underlying application), to create a set of *candidate output* (alignments).
- The advisor estimator is then used to label each of the candidate alignments with an estimated accuracy.
- Finally, using the labeled estimated accuracy the candidate alignment that is predicted to be most accurate is returned to the user, and therefore selecting the parameter vector.

Parameter advising is an *a posteriori* selection method and may not work in all cases, but in those applications where resources can be allocated such that each of the candidate outputs and their labels can be produced in parallel, the advising procedure adds only a negligible amount of wall-clock time (the time needed run the estimator, also in parallel, and select the best output). In addition to multiple sequence alignment, this framework has been shown to work well for selecting parameters for transcript assembly [11].

Throughout the paper it is assumed that there is a consistent set B of benchmark sequence sets (the precise sets themselves will be explained in Section 3.3). Because there is bias in the sampling of these benchmarks (as they are made by hand, additional commentary on this in later sections), it is assumed that at all stages each benchmark will have a weight (or importance), w_b for $b \in B$. This weighting will be used to correct for the overabundance of some types of examples.

2.1 Advisor Sets

For the task of parameter advising an advisor set need to satisfy two main criteria: the set should be small to reduce resource consumption, and should contain at least one parameter vector that will work well on each given input. These two ideas are somewhat contradictory thus a method is needed to balance the competing interests. One way to do this is to put an actual limit on the resource consumption by specifying k, the number of parameter vectors to include in our set. Experiments can be used to determine the cardinality at which there are diminishing returns, meaning adding more computational power does not provide substantial gains in advising accuracy.

Existing methods have been developed to accomplish the task of finding such sets with restricted cardinality. They all rely on first enumerating the entire universe U of parameter vectors for a tool across the entire set of benchmarks B. The combination of a benchmark $b \in B$ and a parameter from the universe $p \in U$ produces an alignment, $\mathbb{A}_{bp}$. Because the reference alignment is known the true accuracy, A_{bp} can be calculated. If there is a pre-existing accuracy estimator it is assumed that the estimated accuracy for each of these alignments is E_{bp}.

The *oracle* set finding method uses only the true accuracy, and finds sets that are optimal if you had an estimator function that predicted accuracy exactly (an oracle). Finding optimal sets while taking the accuracy estimator into account is not practical, thus a *greedy approximation* method was developed that works well in practice.

2.1.1 Oracle Sets

While finding optimal parameter sets is NP-Complete [9] whether you have the estimator values or not. For a small enough universe the Oracle set problem can be solved exactly using an integer linear program (ILP).[1] The program contains variables s_p

[1] As is noted in Section 4 for the much larger universe used in most of the study is it not possible to solve these problems exactly at the time of publication so results are shown on the smaller parameter universe from previous studies for only the Oracle sets.

which indicates if parameter p is in the advisor set, and variables v_{bp} that will signify if the advisor using the found set chooses parameter p for benchmark b. Three groups of constraints then ensure that:
(1) for each benchmark has only one chosen alignment,

$$\sum_{p \in U} v_{bp} = 1$$

(2) an alignment is only chosen if the associated parameter is chosen,

$$v_{bp} \leq s_p$$

and (3) that only k parameters are chosen.

$$\sum_{p \in U} s_p \leq k$$

With the constraints in place, the objective then would be to maximize the total accuracy of all of the chosen alignments

$$\sum_{b \in B} \sum_{p \in U} v_{bp}(w_b A_{bp}).$$

The ILP then contains $O(|U||B|)$ variables and constraints, though only $O(|U|)$ of the variables need to be explicitly declared as binary (integer), if the parameter variables are binary the alignment variables must be binary valued for a solution to be optimal (with the exception of ties in accuracy, but this would not impact optimality).

2.1.2 Greedy Sets

Finding optimal estimator-aware sets for large cardinalities is impractical, but a greedy procedure has been shown to provide substantial accuracy increases (over Oracle sets) in practice. Assume that an advisor $\mathcal{A}_\delta(P)$ takes as input the set of parameter vectors $P \subseteq U$, and returns the average advising accuracy (over all benchmarks in B) of a parameter advisor that uses P as its advisor set (it has access to the set of benchmarks, true accuracies, and estimated accuracies). For generalization purposes during training (but not for showing the results in later sections), a margin of error on the estimator, δ, is used within the function $\mathcal{A}$. Rather than using a single alignment's accuracy for a benchmark (the one with highest estimated accuracy), the method calculates the average for all alignments of a benchmark that are within δ of the maximum (details of this can be found in [9]).

The initial set P_1 is the Oracle set of size 1, in other words the best single default parameter vector. At each step the greedy procedure finds the *next best* parameter to add that is not already in the set, i.e. the one that when added provides the highest advisor accuracy. That is to say the greedy method at iteration i first finds

$$p_i = \operatorname*{arg\,max}_{p' \in U \setminus P_{i-1}} \left\{ \mathcal{A}_\delta \left(P_{i-1} \cup \left\{ p' \right\} \right) \right\},$$

then sets $P_i = P_{i-1} \cup \{p_i\}$. It continues with this procedure until P_k is found.

2.2 Advisor Estimator

The task the advisor estimator must fulfill is to rank the candidate outputs such that the highest-ranked alignment is the most accurate. The best estimator is referred to as the Oracle (also used above) which knows the true accuracy and would be able to make a ranking based on this information. In the absence of an Oracle something close needs to be found. There are in theory two types of advisor "estimators": one that assigns an estimated accuracy to each candidate (as will be used here), and another that simply chooses from the set of candidates. This second type of advisor is yet to be explored, but would still likely need to rely on some way to extract information (features) from the alignment itself in order to make a judgment.

2.2.1 Feature Functions

The feature functions used were heavily studied in previous work, they consist of efficiently-computable functions that extract a singular numeric value that is correlated with accuracy. This set contains both canonical feature functions that have been historically used for benchmarking as well as some developed specifically for this task. Because the focus is on *protein* multiple sequence alignments, many of the feature functions will exploit the predicted secondary structure of the sequences in the alignment to calculate a value. This means each amino acid in the alignment is labeled with both a class (α-helix, β-strand, or coil) and a set of three probabilities which the features utilize.

The features developed are also non-local, meaning they generally use information from across the entire alignment. This means that they contain more information than an alignment objective function (which by design are local calculations).

Details of the feature computations can be found in [17], but they are listed here for convenience as they are a crucial component of our new methods, roughly in order of importance:

- **Secondary Structure Blockiness** — maximum percentage of an alignment that can be covered by a packing of contiguous two-dimensional blocks of residues with the same secondary structure class label.
- **Percent Identity** and **Secondary Structure Percent Identity** — the fraction of aligned amino acids (structure class labels) in all extracted pairwise alignments from the input that are precisely the same.
- **Average Replacement Score** — average scaled BLOSUM62 score for all aligned amino acids from the induced pairwise alignments.
- **Gap Extension Percentage** and **Gap Open Percentage** — percentage of characters in the alignment that are (runs of) gap characters.
- **Information Content** — average over all columns of the difference in distribution of amino acid frequencies compared to a background distribution.
- **Substitution Compatibility** and **Gap Compatibility** — fraction of pairs of columns that pass the 4 gametes test when converted into 1 and 0 by use of majority/minority amino acid character or gap/non-gap character.
- **Structure Agreement** — weighted sum of structure class probabilities for a window around each aligned pair of amino acids in induced pairwise alignments.
- **Gap/Coil Density** — fraction of gap characters that align with the coil structure class label.
- **Core Column Density** — fraction of columns that are gap-free and contain mostly the same amino acid.
- **Sequence Consensus** and **Gap Consensus** — fraction of pairs of columns that have the same majority/minority amino acid character or gap/non-gap character pattern.

All feature values are normalized to be in the range $[0, 1]$, and if the value cannot be defined (i.e. gap open percentage when there are no gaps) the value is fixed at 0.

2.2.2 The `Facet` Estimator

The original `Facet` (short for Feature-based Accuracy Estimator) model uses a linear combination of the feature values to calculate an accuracy estimate. Given a set of weights $T = (t_1, t_2, ..., t_n)$ the `Facet` value is computed from a set of feature functions $F_1, F_2, ..., F_n$ as

$$L_T(\mathbb{A}) := \sum_{1\leq i\leq n} t_i F_i(\mathbb{A}).$$

There are several methods that can be used to learn an accuracy estimator, one can either try to match the estimator values to the true accuracy exactly or, what worked better in previous experiments for the task of parameter advising for multiple sequence alignment, focus on the objective of ranking candidates.

All of the original `Facet` results shown will use the methods described in [10], which learns the values of T using combinatorial optimization (in this case linear programming, LP). In this case for all pairs of alignments $\mathbb{A}_{bp}$ and $\mathbb{A}_{bq}$ aligning the same benchmark using parameters p and q such that $A_{bp} > A_{bq}$, the goal is to minimize the ranking error:

$$\max\left\{\left(\left(A_{bp} - A_{bq}\right) - \left(L_T\left(\mathbb{A}_{bp}\right) - L_T\left(\mathbb{A}_{bq}\right)\right)\right), 0\right\}.$$

Note that in this case, the only free variables are T, because the alignments are fixed then the feature values and true accuracies are also. Additional constraints are added to ensure that the estimator as well as the values in T are in the range $[0, 1]$.

3 Modern Machine Learning for Accuracy Estimation

3.1 `Facet-NN`: Exploiting Non-Linearity

3.1.1 Learning an architecture

While network architectures are sometimes inspired by the underlying structure of the input data [23], when such inspiration is not viable, as is the case here, a neural network architecture can be found empirically using guided search. Using an architecture

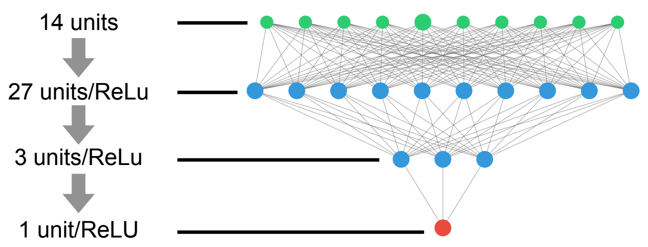

Figure 2: Cartoon of architecture learned for one cross-validation fold

with too many parameters is time-consuming when training and requires more training data. This extended resource consumption makes exploring the set of architectures with large numbers of parameters computationally expensive. Therefore, like most works, certain architectures are excluded from this search space that are likely to have poor performance both resource-wise given their training time and accuracy-wise due to the fixed set of training examples.

The hyperparameter optimization procedure used (from the KerasTuner package [26]) required defining a domain of possible hyperparameter values and sampling random points from that domain based on a random search. Thus the main task is then to develop a search space rather than a model. Then using a randomized sampling technique the network architecture with the lowest mean-squared error can be found.

The sampled hyperparameter universe consisted of three main classes: the number of neurons for each hidden layer, the number of hidden layers, and the activation functions for each neuron. The final hyperparameter universe was as follows: 3, 9, and 27 neurons in each layer, between 1 and 3 hidden layers, and either ReLu and Sigmoid activation functions selected independently for each layer. The procedure was repeated independently for each cross-validation fold.

As an example, Figure 2 shows the network used on a single cross-validation fold (note that hidden layer 1 is not drawn to scale for readability). After 20 iterations the final architecture was a neural network of 2 hidden layers, the first layer with 27 neurons and with ReLu as their activation function, and 3 neurons for the second layer with ReLu as their activation function. This network has a total of 493 trainable weights.

3.1.2 Learning a model

Once an architecture was found for each cross-validation fold, weights for the model could be learned using the adaptive gradient algorithm (via the `AdaGrad` optimizer [12]). One issue is that, as was alluded to in Section 2, the training benchmark sequence sets are skewed toward easy-to-align protein groups. This runs in contrast to the underlying assumption of most machine learning algorithms which is that the data is equally distributed among observations. With imbalanced datasets, the algorithm tends to skew its predictions toward the majority class (the accuracies within the class of near-perfect alignment accuracy in our case). Without enough data, it is impossible for the algorithm to learn the patterns of the minority class (poor alignment accuracy).

As was done when training `Facet`, in `Facet-NN` the samples are weighted to account for bias and minimize the impact of skewed data on our training. Weights are assigned to the samples based on the number of observations per bin as described below, but here the bin is not based on the accuracy of the alignment produced using the default parameter vector but the alignment itself. That is, weights are not determined by the underlying *benchmark* sequence set but by the *alignment* accuracy. Alignments are assigned evenly to n bins, spread evenly across the range of accuracies, and a weight was assigned based on the following formula using the balanced sample weighting in `scikits-learn` [27]:

$$\frac{m}{(n * C(A_{bp}))},$$

where m is the total number of alignments in the training set and $C(A_{bp})$ gives the number of other alignments in the same bin as alignment A_{bp}.

A note about implementation

All of the methods above as well as the experiments below were performed using GPU-enabled `TensorFlow` [1] in combination with the `Keras` [7] package. All models, architectures, and training scripts (written for `python3` [33]) are for both `Facet-NN` and `Facet-LR` are released on the DeBlasio Lab GitHub, at `github.com/deblasiolab/Facet-NN`.

3.2 `Facet-LR`: Utilizing More Data

As mentioned in Section 2.2.2, `Facet` was trained to minimize the ranking error of the accuracy prediction made by the tool. This was done mainly because it showed better performance than trying to match the accuracy value directly. One reason this was likely the case is that the LP method used was not able to scale to large numbers of examples practically, and thus narrowing the task to one close to the actual use was better with that little data. But, modern machine learning optimization tools now allow us to use much more data to train models, they also enable us to fully utilize complex hardware to accomplish this task.

`Facet-LR` was implemented using the Linear Regression model of `scikits-learn`. A separate estimator was learned on the training set of each cross-validation fold, minimizing the residual sum of squares between the predicted and true accuracy of each alignment. All results are reported as an average over all of the folds.

As an example `Facet-LR` trained on a single fold is calculated using the weights in Table 1, sorted by the absolute value of their influence. Note that some of the features have negative weights, this was not allowed in the original `Facet`. While not intuitive, since all of the features originally trended with accuracy, it can be explained: for instance if an alignment has a high sequence identity but also a high gap density, these two values (because they have opposite signs) will counteract each other. More of the features in `Facet-LR` have significant weight compared with `Facet`, in the previous studies only 5 features ended up being used while here more than 9 features make contributions to the final score.

Table 1: Example `Facet-LR` Feature Weights

Feature Function	**Weight**
Secondary Structure Percent Identity	1.232
Gap Open Percentage	−0.551
Secondary Structure Blockiness	0.475
Average Replacement Score	−0.455
Core Column Density	−0.311
Substitution Compatibility	−0.296
Gap Compatibility	−0.260
Gap Consensus	−0.147
Structure Agreement	−0.113
Gap/Coil Density	−0.072
Information Content	0.052
Percent Identity	−0.049
Gap Extension Percentage	0.015
Sequence Consensus	0.005

3.3 An Expanded Parameter Universe

Two *universes* of parameter vectors are used: one containing just over 2000 parameter vectors used for the original `Facet` called "original", and an extended set of 16,896 parameter vectors called the "extended" universe. The original set was used to construct all of the Oracle advisor sets as well as the original `Facet` estimator, the extended set was used for training `Facet-NN` and for the Greedy advisor sets.

The extended set was originally developed for use in the original `Facet` methodology. It is constructed by first selecting 8 commonly use high-accuracy replacement matrices (`VTML20`, `40`, `80`, `120`, `200`, and `BLOSUM45`, `62`, `80`). For each of the four `Opal` gap penalty parameters (gap-open and gap-extension for both terminal and non-terminal gaps) several possible values were enumerated, and the cross product of these discrete values and the replacement matrices produced a collection (universe) of parameter vectors. Using the default parameter vector as a starting (which was found via inverse parametric alignment [20, 18]) we sampled up to five additional values for each parameter above and below the default. In the case of the terminal gaps, a value relative to the corresponding non-terminal parameter value was chosen rather than a specific penalty. This set constituted approximately 2,000 choices of gap penalty combinations, leading to the number of parameter vectors above in our extended universe.

The original universe is then an informed subsampling of this universe, used to allow the older methods to be solved.

Building on the datasets used in [10], a carefully curated set of 861 benchmark sequence sets consisting of examples from PALI [4] and BENCH [14] (which is itself a collection of alignments from OxBench [29], SABRE [34], and BAliBase [3]) is used here. Each benchmark consists of a set of protein sequences as well as a reference alignment which is generally induced using an alignment of the known underlying three-dimensional protein structures. The alignment information is removed to also retrieve the set of constituent sequences. Because the reference is known, the accuracy can be determined as defined earlier.

As mentioned in Section 2 the set of benchmarks is biased toward easy-to-align sets of sequences. This is likely because these benchmark alignments are constructed mostly by hand, and thus only the alignments with which they have the most confidence end up being released. But, in practice, it is expected that sequence sets from across all levels of difficulty will be input. To correct this, each benchmark's ease of alignment was classified using its accuracy using the `Opal` aligner's default parameter vector. This parameter vector was chosen specifically to work best *on average*, and it is the set of parameter values most users will use. The range of accuracies was then divided into 10 equally sized bins and assigned a benchmark to the bin that corresponds to its accuracy when aligned using the default parameter vector. To exemplify the bias in the sample, note that there is a more than 36x increase in the size of the easiest-to-align bin (those with accuracy above 90%, 434) and the hardest-to-align (those $\leq$ 10%, 12). Unless otherwise noted, the weights w_b will be assigned to evenly distribute among *bins* rather than benchmarks. This means with this scheme the accuracy of using only the default parameter vector will be close to 50%, rather than near 80% when weighting by benchmark.

In all cases, 12-fold cross-validation is implemented to divide the data into training and testing sets. Due to the binning described above, this number of folds is somewhat forced since the smallest bins have 12 benchmarks a piece. In each case, each bin was divided randomly into 12 sets, then one set from each bin was used in each cross-validation fold to be part of the test (hold-out) set while the others were used for training and validation. All of the results below show the average across all 12 cross-validation folds.

4 Experimental Results

4.1 Accuracy of the Estimator

Figure 3 shows the comparison of the estimated value (vertical) with the true accuracy (horizontal) for each of the 14 million alignments in the dataset across the two estimators. Each alignment is shown for when the benchmark it is aligning is in the testing set. To work well as an advisor estimator the value should trend with true accuracy and the plot

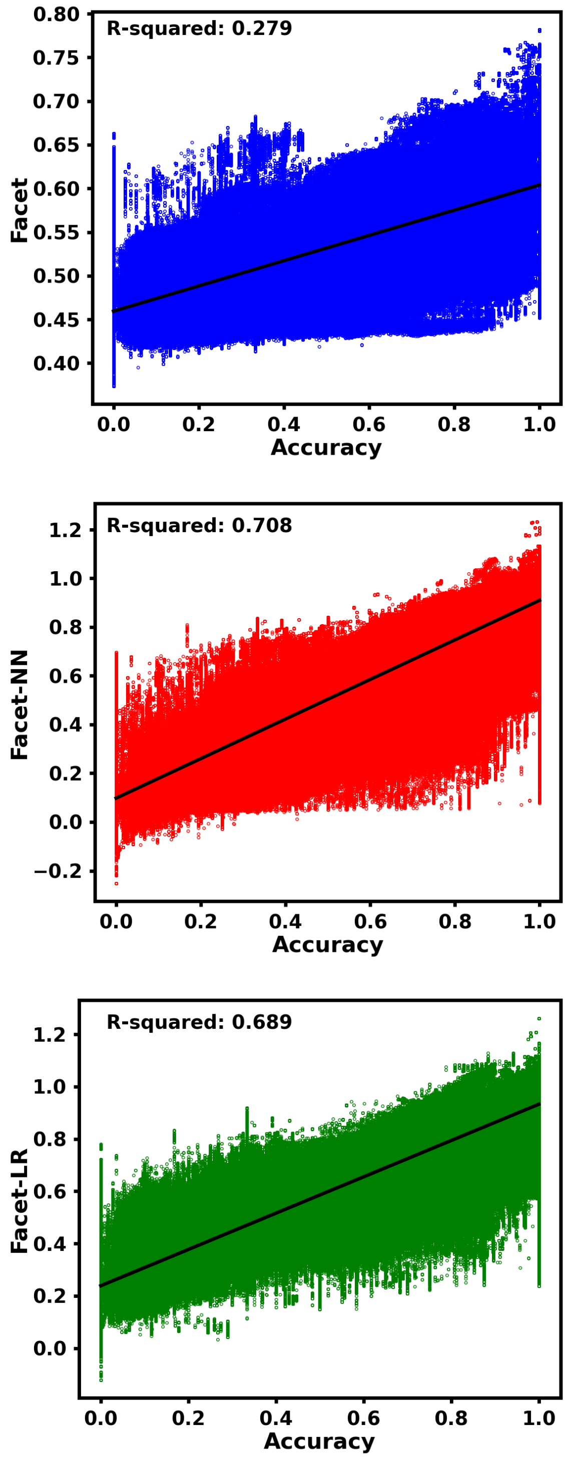

Figure 3: Estimated versus true accuracy of all alignments in the dataset for `Facet`, `Facet-NN`, and `Facet-LR`.

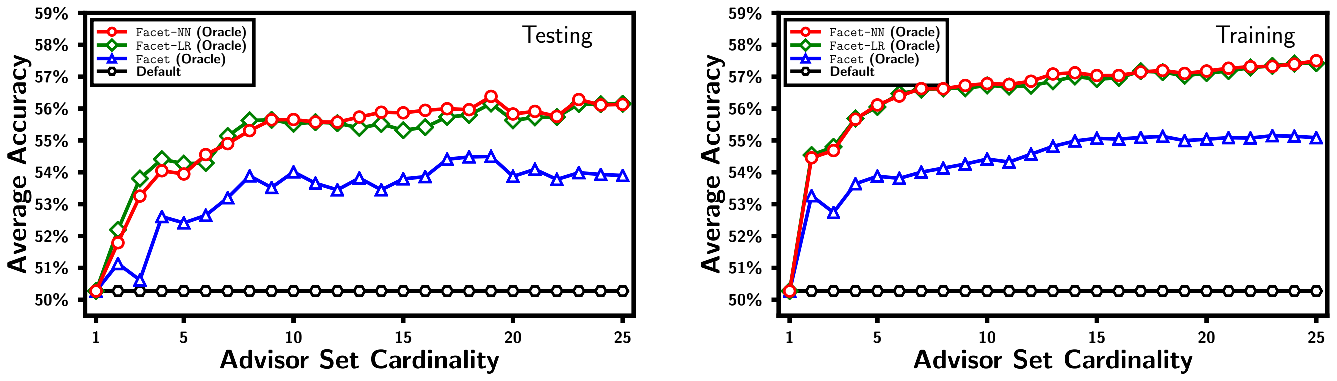

Figure 4: Comparison of the advising accuracy of `Facet`, `Facet-NN`, and `Facet-LR` using Oracle advisor sets

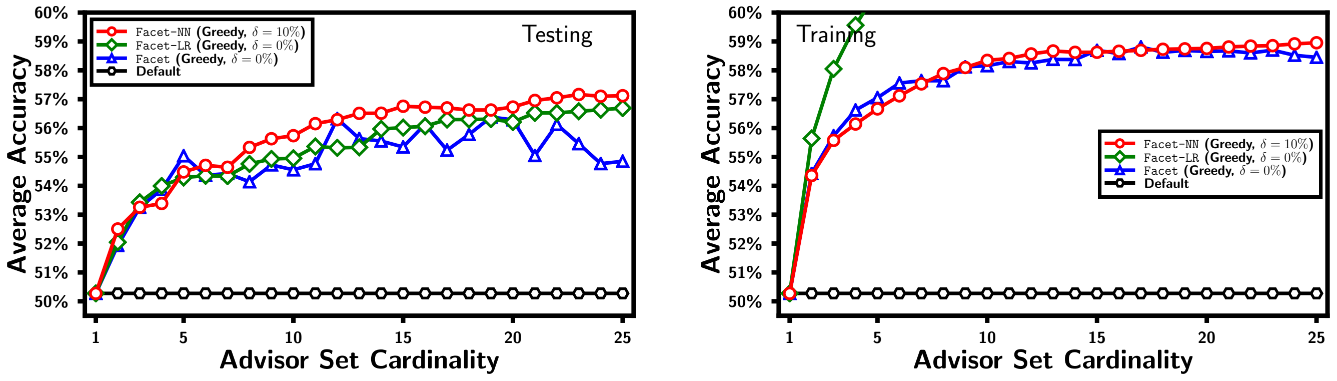

Figure 5: Comparison of the advising accuracy of `Facet` and `Facet-NN` using Greedy advisor sets

should have high slope and low spread in order to correctly distinguish between candidates. Facet-NN and Facet-LR have both a steeper slope and less spread than Facet. The trend-line is shown that is the best linear fit to all of the alignments plotted. The conclusion that Facet-NN and Facet-LR outperform Facet on the task of matching accuracy is also corroborated by the R-squared values shown, Facet-NN has a value of 0.708 and Facet-LR 0.689 while original Facet is only 0.279. The Facet summary R-Squared is very much impacted by the variability in the learned estimators, with a mean of 0.631 across the 12 cross-validation folds compared to 0.721 of Facet-NN and 0.691 of Facet-LR. On every cross-validation fold, Facet-NN shows an increase in R-squared, with an average increase of 0.090, and Facet-LR an increase of 0.060.

One contributing factor to the difference seen in the correlation of the estimator and accuracy is that due to the computational limitations at the time, the Facet estimator is learned over only parameter pairs in the Oracle set of a given size (in the case of the results above cardinality 25), while because Facet-NN and Facet-LR are able to take advantage of modern GPUs, the whole training set was able to be exploited. This change in the training set used in Facet also means there is variation in the learned weights at different cardinalities as well as between folds (which will be shown in later sections). As mentioned in Section 3.1 unlike original Facet, Facet-NN and Facet-LR were trained to match the accuracy of a given alignment exactly. Therefore, the objective of the Facet-NN and Facet-LR estimators is toward increasing R-Squared.

One other item to note in Figure 3 is the scale on the 3 plots. Because there is nothing tying the estimator values of Facet to the true accuracies the range of values ends up being quite small. In contrast, by training to the actual accuracy we see estimated accuracies across the whole range from 0 to 1; but this comes at the expense of having predicted values that are somewhat contrary to the definition of accuracy, as you cannot recover fewer than 0 (or more than all) pairs of residues.

4.2 Accuracy of the Advisor

The main purpose of developing Facet-NN and Facet-LR was to improve accuracy of the resulting parameter advisor. In the experiments below Facet-NN, Facet-LR, and Facet are used as the accuracy estimator to construct several advisors. In each case, the increase in accuracy gained by using our new estimators at various advisor set sizes is shown. Advisor-oblivious Oracle sets (using the original universe) and advisor-aware Greedy sets (using the extended universe) are used in combination with the two advisors to create 4 classes of advisors. The original Facet method relies on a parameter vector set to learn its weights, thus, in all experiments, the Facet estimator used was trained on the Oracle set of the specified cardinality, whereas only a single Facet-NN and Facet-LR advisor is used for each cross-validation fold.

4.2.1 Oracle Sets

Figure 4 shows the accuracy of the three advisors using Oracle sets on the original universe of parameter vectors as well as accuracy of the default parameter vector. The two plots show the use of the advisor averaged across the 12 cross-validation folds' training (right) and testing (left) benchmarks. Each point on the four lines represents one advisor with its position corresponding to the advisor set cardinality (horizontal) and the advising accuracy (vertical).

Facet-NN and Facet-LR both show a higher advising accuracy across all cardinalities both on the training and testing benchmarks. When using Oracle advisor sets on the testing benchmarks Facet-NN shows an average increase of 1.8% accuracy, and Facet-LR is similar. Because Facet-NN and Facet-LR are trained on a much larger number of parameters as mentioned above, the results show less variation as you increase advisor set cardinality; while not perfectly monotonic it shows less variation than Facet. Also, while both accuracy estimation tools eventually plateau (showing little improvement in accuracy with increased cardinality) Facet-NN and Facet-LR seem to do this at a larger set size.

With respect to the training benchmarks, Facet is trained on only the pairs of alignments from the Oracle set being shown. Previous results had shown that fitting Facet to differences in accuracy showed higher advising accuracy, but when a more accurate estimator like Facet-NN or Facet-LR is used there is an improvement even over Facet which was specifically designed for this task.

4.2.2 Greedy Sets

Figure 5 shows the accuracy of advisors using Greedy advisor sets trained specifically for the individual estimators. Once again, the two plots show the use of the advisor averaged across the 12 cross-validation folds' training (right) and testing (left) benchmarks. Just as above, each point on the four lines represents one advisor with its position corresponding to the advisor set cardinality (horizontal) and the advising accuracy (vertical).

`Facet-NN` nor `Facet-LR` benefit as much from Greedy sets as the original `Facet` with respect to the training benchmarks, seeing an average increase of 1.3% compared to using Oracle sets. That said, while at small cardinalities `Facet-NN`, `Facet-LR`, and `Facet` show similar performance, across the cardinalities shown the average increase in advising accuracy is still 2.2%. This is mainly due to the fact that the `Facet` estimator is inconsistent in its accuracy as you vary the set cardinality; note that the curve shown here has Greedy sets that use the larger extended parameter universe, but this was seen on the smaller universe as well.

On the testing benchmarks, all three advisors show similar performance, but it seems that because of the higher generalization value used for `Facet-NN` the drop in accuracy between training and testing is smaller. Various values of δs were tested for `Facet-NN` and `Facet-LR`, the most accurate value is shown here (for `Facet` $\delta = 0$ was shown to be superior in previous studies). Note that the training curve for `Facet-LR` is cut off in Figure 5 for readability, but can be seen in its entirety in Figure 7, this large drop in accuracy is due to the chosen value of δ.

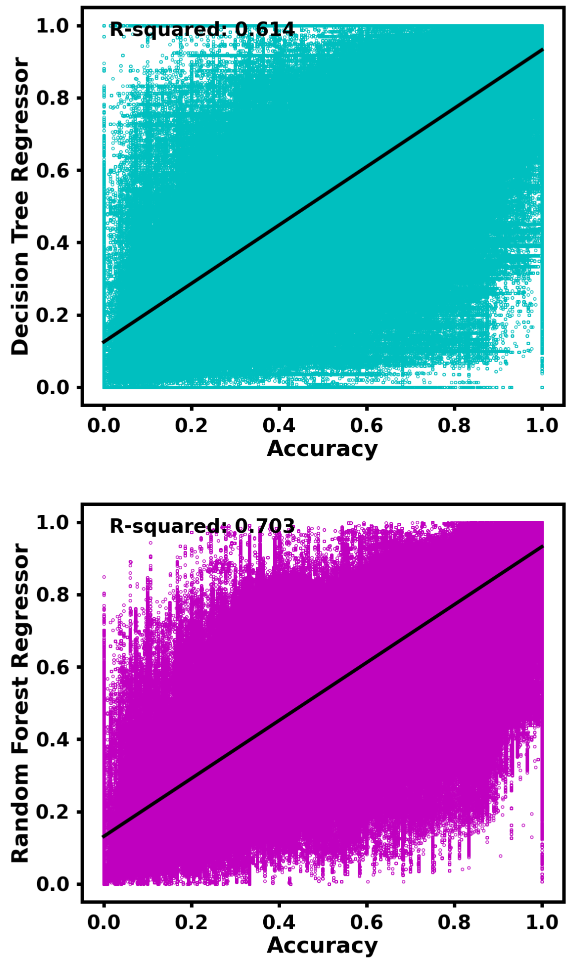

Figure 6: Estimated versus true accuracy of all alignments in the dataset for other machine learning models.

4.3 Comparison of Models

It is critical to choose the right algorithm to train a model. To compare the results of our neural network with those of other popular machine learning algorithms, models were trained using decision tree regression and random forest regression. The estimated accuracies of these three models are shown in Figure 6. Both `Facet-NN` and `Facet-LR` have a higher correlation (R-squared) with true accuracy than any of the other models on the testing set. However, both of the other models showed an almost perfect correlation with the training data.

The estimators produced with the other machine

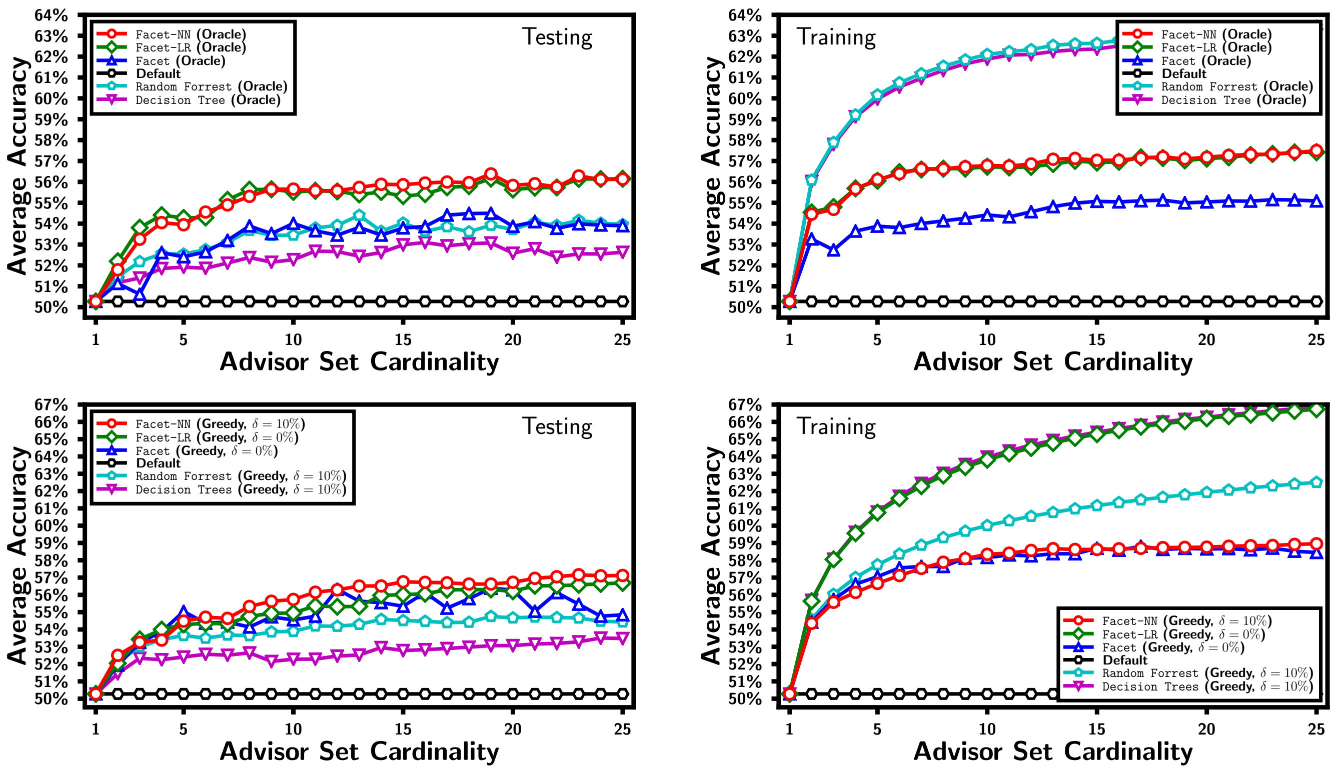

Figure 7: Comparison of the advising accuracy of the other regressors

learning methods were also tested as advisor estimators. The results are shown in Figure 7. When this experiment is performed we see just how much the decision tree and random forest regressors overfit to the training benchmarks. On both the Oracle and Greedy advising sets, they have lower performance than `Facet` (and in turn `Facet-NN` and `Facet-LR`) on the testing set while having training performances that are significantly higher than the other methods.

5 Conclusion

When estimating the accuracy of protein multiple sequence alignments a major improvement can be obtained by exploiting modern machine learning tools. Previous studies have shown that when estimating accuracy, the features that need to be extracted in order to create an estimation for alignments of any size are very important. While using carefully constructed small sets of parameter vectors for training can have very good performance as an accuracy estimator for parameter advising, using large amounts of data and models that can be trained on it shows even better accuracy using the same feature functions. When using the old training methods, along with advisor-agnostic advisor set, you can get an increase of about 4% over using the default parameter vector alone, an increase of 6% is seen when using `Facet-NN`. Additionally, by training the estimator over the entire set of parameter vectors `Facet-NN` and `Facet-LR` shows more stability between cross-validation folds and is more generalizable.

5.1 Ongoing Work

Due to time constraints, we were unable to get results using other estimators (such as TCS [5] or MOS [22]) on the larger parameter universe before the time of submission. We do plan to include these in any follow-up studies. But, previous studies have shown that for the task of advising, `Facet` is far superior, therefore we are confident that `Facet-NN` and `Facet-LR` will also outperform them.

We also continue to explore new architectures, while the use of fully connected networks yield a satisfactory performance, some more advanced structures such as residual networks [6] are currently being explored.

While it should technically be possible, as formulated, finding Oracle Sets on the extended universe is currently intractable. As described in previous studies, the method uses an integer linear program, which was solvable in the original universe despite being technically NP-Complete. When constructed for the extended universe we were unable to find a solution within a week using either CPLEX or Gurobi (the two commonly used tools) on a machine with 256 threads and 2 terabytes of memory. Finding these sets would allow us to use the same universe at all stages of this study.

Acknowledgements

The authors would like to thank Dr. Olac Fuentes and Jose Perez for fruitful discussions and the UTEP Campus Office of Undergraduate Research Initiatives (COURI). This work was partially funded by the Computing Alliance of Hispanic-Serving Institutions (CAHSI) Research Experience for Undergraduates (REU) under NSF Grant Numbers CNS-2137791 and HRD-1834620. Additional funding was provided by the US Department of Education project number 226150864A.

References

[1] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. `https://www.tensorflow.org/`

[2] Virpi Ahola, Tero Aittokallio, Mauno Vihinen, and Esa Uusipaikka. 2008. Model-based prediction of sequence alignment quality. *Bioinformatics* 24, 19 (Sept. 2008), 2165–2171.

[3] Anne Bahr, Julie D Thompson, J C Thierry, and Olivier Poch. 2001. `BAliBASE` (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. *Nucleic Acids Research* 29, 1 (Jan. 2001), 323–326.

[4] S Balaji, S Sujatha, S Sai Chetan Kumar, and N Srinivasan. 2001. `PALI`: a database of Phylogeny and ALIgnment of homologous protein structures. *Nucleic Acids Research* 29, 1 (Jan. 2001), 61–65.

[5] J M Chang, P D Tommaso, and Cedric Notredame. 2014. `TCS`: a new multiple sequence alignment reliability measure to estimate alignment accuracy and improve phylogenetic tree reconstruction. *Molecular Biology and Evolution* 31, 6 (April 2014), 1625–1637.

[6] Dongwei Chen, Fei Hu, Guokui Nian, and Tiantian Yang. 2020. Deep Residual Learning for Nonlinear Regression. *Entropy* 22, 2 (2020), 1–14. `https://doi.org/10.3390/e22020193`

[7] Francois Chollet et al. 2015. *Keras*. `https://github.com/fchollet/keras`

[8] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. 1978. A model of evolutionary change in proteins. *In Atlas of Protein Sequences and Structure* 5 (1978), 345–352.

[9] Dan DeBlasio and John Kececioglu. 2017. Learning Parameter-Advising Sets for Multiple Sequence Alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 14, 9 (2017), 1024–1041. Online 2015.

[10] Dan DeBlasio and John Kececioglu. 2018. *Parameter advising for multiple sequence alignment.* Springer International Publishing.

[11] Dan DeBlasio, Kwanho Kim, and Carl Kingsford. 2020. More Accurate Transcript Assembly via Parameter Advising. *Journal of Computational Biology* 27, 8 (2020), 1181–1189. `https://doi.org/10.1089/cmb.2019.0286`

[12] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, 7 (2011), 2121–2159.

[13] Robert C. Edgar. 2004. `MUSCLE`: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research* 32, 5 (2004), 1792–1797.

[14] R C Edgar. 2009. `BENCH`. `http://www.drive5.com/bench`.

[15] S Henikoff and J G Henikoff. 1992. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences USA* 89, 22 (Nov. 1992), 10915–10919.

[16] K Katoh, Kei-ichi Kuma, Hiroyuki Toh, and Takashi Miyata. 2005. `Mafft` version 5: improvement in accuracy of multiple sequence

alignment. *Nucleic Acids Research* 33, 2 (Jan. 2005), 511–518.

[17] John Kececioglu and Dan DeBlasio. 2013. Accuracy estimation and parameter advising for protein multiple sequence alignment. *Journal of Computational Biology* 20, 4 (April 2013), 259–279.

[18] John Kececioglu and Eagu Kim. 2006. Simple and Fast Inverse Alignment. In *Proceedings of the 10th Conference on Research in Computational Molecular Biology (RECOMB)*. Springer-Verlag LNBI 7262, 441–455.

[19] John Kececioglu and Dean Starrett. 2004. Aligning alignments exactly. In *Proceedings of the 8th Conference on Research in Computational Molecular Biology (RECOMB)*. ACM, 85–96.

[20] Eagu Kim and John Kececioglu. 2008. Learning Scoring Schemes for Sequence Alignment from Partial Examples. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 5, 4 (2008), 546–556.

[21] Giddy Landan and Dan Graur. 2007. Heads or tails: a simple reliability check for multiple sequence alignments. *Molecular Biology and Evolution* 24, 6 (2007), 1380–1383.

[22] Timo Lassmann and Erik L L Sonnhammer. 2005. Automatic assessment of alignment quality. *Nucleic Acids Research* 33, 22 (Dec. 2005), 7120–7128.

[23] Jianzhu Ma, Michael Ku Yu, Samson Fong, Keiichiro Ono, Eric Sage, Barry Demchak, Roded Sharan, and Trey Ideker. 2018. Using deep learning to model the hierarchical structure and function of a cell. *Nature Methods* 15 (April 2018), 290–298.

[24] Tobias Müller, Rainer Spang, and Martin Vingron. 2002. Estimating amino acid substitution models: a comparison of Dayhoff's estimator, the resolvent approach and a maximum likelihood method. *Molecular Biology and Evolution* 19, 1 (Jan. 2002), 8–13.

[25] Cedric Notredame, Desmond G Higgins, and J Heringa. 2000. `T-Coffee`: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology* 302, 1 (Sept. 2000), 205–217.

[26] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. 2019. KerasTuner. `https://github.com/keras-team/keras-tuner`.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[28] O Penn, Eyal Privman, H Ashkenazy, Giddy Landan, Dan Graur, and Tal Pupko. 2010. `GUIDANCE`: a web server for assessing alignment confidence scores. *Nucleic Acids Research* 38, Web Server (June 2010), W23–W28.

[29] GPS Raghava, Stephen MJ Searle, Patrick C. Audley, Jonathan D. Barber, and Geoffrey J. Barton. 2003. `OXBench`: A benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics* 4, 1 (2003), 1–23.

[30] F Sievers et al. 2011. Fast, scalable generation of high-quality protein multiple sequence alignments using `Clustal Omega`. *Molecular Systems Biology* 7, 1 (Jan. 2011), 539–539.

[31] J D Thompson, Desmond G Higgins, and Toby J Gibson. 1994. `Clustal W`: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research* 22, 22 (1994), 4673–4680.

[32] J D Thompson, Frédéric Plewniak, Raymond Ripp, Jean-Claude Thierry, and Olivier Poch. 2001. Towards a reliable objective function for multiple sequence alignments. *Journal of Molecular Biology* 314, 4 (Dec. 2001), 937–951.

[33] Guido Van Rossum and Fred L. Drake. 2009. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[34] Ivo Van Walle, Ignace Lasters, and Lode Wyns. 2005. `SABmark`: a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics* 21, 7 (March 2005), 1267–1268.

[35] L Wang and T Jiang. 1994. On the complexity of multiple sequence alignment. *Journal of Computational Biology : a Journal of Computational Molecular Cell Biology* 1, 4 (1994), 337–348.

[36] Travis J Wheeler and John D Kececioglu. 2007. Multiple alignment by aligning alignments. *Proceedings of the 15th ISCB Conference on Intelligent Systems for Molecular Biology (ISMB), Bioinformatics* 23, 13 (July 2007), i559–i568.

[37] Willard I. Zangwill. 1969. *Nonlinear Programming: A Unified Approach.* Prentice-Hall International, Englewood Cliffs, N.J.