ESCI upf. School of International Studies

MEMÒRIA DEL TREBALL DE FI DE GRAU DEL GRAU (ESCI-UPF)

Optimization of alignment preprocessing used as input on a residual neural network for substitution model selection

AUTOR/A: Antón Vega Méndez	**NIA:** 104530
GRAU: Bachelor's Degree in Bioinformatics	
CURS ACADÈMIC: 2021-2022	
DATA: 21-6-2022	
TUTOR/S: Sebastian Burgstaller-Muehlbacher	

Optimization of alignment preprocessing used as input on a residual neural network for substitution model selection

Antón Vega Méndez

Academic tutor: Sebastian Burgstaller-Muehlbacher[1,2]

[1] *Center for Integrative Bioinformatics Vienna, Max Perutz Labs, University of Vienna*
[2] *Medical University of Vienna, Vienna BioCenter (VBC), Vienna, Austria*

When performing phylogenetic tree reconstruction, we often see that the use of an incorrect model of sequence evolution can lead to a wrongly modeled tree. However, even though choosing the right model of sequence evolution is the first step for phylogenetic tree reconstruction, using established methods can be computationally expensive and prone to errors.
ModelRevelator[3] uses deep learning to decide which phylogenetic model to use for a certain multiple sequence alignment analysis. In this project, we tried to improve the performance of one of ModelRevelator's networks by using a different approach to preprocessing the information contained in multiple sequence alignment: By randomly selecting subsets of the alignments we computed a set of summary statistics to summarize the information contained in each of those alignments. The end goal was to see whether this approach yielded better results than the previously ones used by ModelRevelator.

1 Introduction

With the increasing number of biological data available nowadays, one of the best ways for phylogeneticists to study evolution and to study the relationship between organisms is to do so through their genetic sequences.

One key step for carrying out this analysis, and when trying to reconstruct an evolutionary tree, is computing the genetic distance, which is a measure of the divergence between two sequences that come from the same common ancestor.

This is a highly challenging task, since many evolutionary forces are applied to the sequences making them vary, which exponentially increases the complexity of comparing them[16]. There are several of these forces and reasons for mutations in the sequences occurring, and we must understand as many as we can to carry our analysis. For example, Transitions and transversions are the two types of these nucleotide mutations. Transitions account for the substitution between two purines or two pyrimidines (A to G or C to G interchangeably), while transversions are the substitution between a pyrimidine and a purine (G to C, A to T...). We distinguish between these two types of substitutions because, although there are many more probabilities of having a transversion, there are higher possibilities of having a transition, due to the chemical structure of the nucleotides as seen in figure 1.

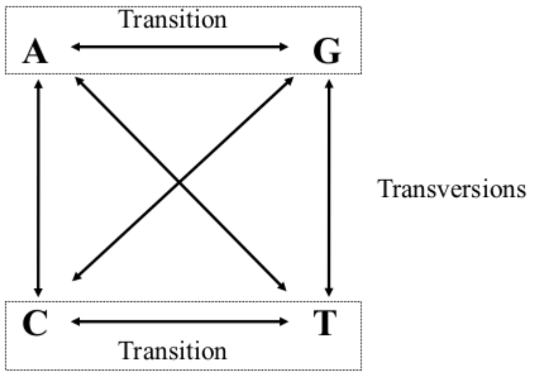


Figure 1: *Diagram showcasing the difference between transitions and transversions*

One of the main requirements for computing the genetic distance is estimating a model of sequence evolution. They use continuous-time Markov-Chains to model the previously mentioned evolutionary forces, and to help describe the different probabilities of change in the sequences across time by making assumptions of the substitutions. One of these assumptions is, for example, the homogeneous frequency of nucleotides that the Jukes and Cantor[9] model takes: This model assumes that there is the same probability for each nucleotide to be on the sequence, as opposed to the GTR[24] model where the frequency of each nucleotide can be different across the sequence.

1.1 Models of sequence evolution

There are hundreds of models to choose from, but we will only account for 6 of them in this project which are widely used and range in complexity; the project can easily be expanded to more in the future:

- **Jukes and Cantor (JC[9]):** This is the simplest substitution model, because as mentioned before it assumes same frequency of nucleotides in the sequence at ¼, and each of those is equally likely to be replaced by any other nucleotide.
- **Kimura 2-parameter (K2P[12]):** Introduces transition and transversion substitution rates to the already mentioned JC parameters.
- **Felsenstein-81 (F81[4]):** Has the same parameters as JC, but it allows the possibility of having different base frequencies than the ¼ used in JC and K2P.
- **Hasegawa, Kishino and Yano (HKY[7]):** Introduces transition and transversion substitution rates whilst allowing different base frequencies.
- **Tamura and Nei (TN93[22]):** It distinguishes between two different types of transition (AG and CT) but only one type of transversions. It also assumes different base frequencies.
- **Generalized time-reversible model (GTR[24]):** Has each of the base frequencies of the nucleotides as parameters, as well as each of the 6 transition rates (A to C is the same as C to A).

Although the concept of a model of sequence evolution is relatively straightforward, finding the correct model of sequence evolution within a certain multiple sequence alignment is a challenging task.

1.2 Choosing the model

Choosing a wrong model for our data can severely harm the results of a phylogenetic analysis. However, choosing the model that better fits our data is far from trivial. Since the cost of sequencing is consistently decreasing every year[21], the number of alignments to analyze is getting substantially larger. This fact coupled with the existence of hundreds of different models of evolution makes the process of choosing one a very time consuming and computationally expensive procedure.

There are several statistical techniques that have been developed for choosing the best fitting model, one of which using likelihood ratio tests: The log likelihoods of two models we want to compare the fit of are computed, and usually these two models are nested, meaning that there is only one parameter that is assumed to be different in both models[16]. Usually the comparison is made between all of the models and its parameters, arriving at the best fitting one. This approach, however, has some limitations, for instance it assumes that one of the models being compared has to be correct, which might not be the case. Another method is Bayes factor, which works similarly to the log likelihood but using reversible jump MCMC[6] for the computations. The most common approach to which we will be comparing the results is Maximum Likelihood[10], which uses either the Bayesian Information Criterium (BIC) or the Akaike Information Criterium[17].

In any case, it is safe to assume that these procedures are computationally very intensive.

1.3 Machine learning

Machine learning uses data to create models that help recognize patterns, to classify and predict. There are several ways that machine learning does this, but in this project we will focus on neural networks and deep learning.

A neural network is a set of linear algebra operations that uses certain mathematical operations to the input data so as to retrieve a desired output. It has three parts: The input layer to which the data is given, the hidden layers on which the operations are performed, and the output layer that yields the result of the operations. For this network to "learn" and improve, we follow two different steps: forward and backward propagation.

We shall now explain how a neural network works with all of these concepts together: first, we randomly initialize weights. These weights are values which we multiply our data with, and will be updated using an optimization algorithm in the backward propagation step. Then for each hidden layer we apply an activation function after the input data has been multiplied by the weight, which induces non-linearity to differentiate between the relevant and non-relevant weights.

If there are additional hidden layers following the first one, a similar procedure is carried out eventually resulting in the output for the output layer. This would be the forward propagation step. Following that, the expected output from our network would be compared to the ground truth we were expecting to get, which is called the error. To compute this error, we use a loss function (in our case we use categorical cross entropy, which is used for multi-class classification). Once this error is computed, we use it for a gradient descent method to minimize it, and to be able to update the weights. This is back propagation[19].

All these steps are part of one iteration the network does, called epoch. We also have hyperparameters, that are fixed parameters defined before training, like the batch size (number of training examples shown to the network at the same time in order to have a smoother gradient descent) and the learning rate (step size of how much gradient should be modified towards a minimum).

Usually, for a complex set of data more than one hidden layer is needed for a successful neural network architecture; and in most cases the more of these layers there are, the higher accuracy the network can achieve. The term used for describing a network that uses many of these layers is a deep neural network, which is called Deep learning[14].

There are several applications of machine learning and deep learning in biology that have proven to be a successful strategy to solve biological problems[15][23][11][20].

1.4 ModelRevelator

ModelRevelator[3] uses two multi-layered neural networks. The first one, NNmodelfind, selects the best model of sequence evolution based on multiple sequence alignments. The second one, NNalphafind, computes the alpha parameter if the multiple sequence alignment benefits from incorporating a gamma distribution to account for the rate heterogeneity across sites[26].

Phylogeneticists can input their multiple sequence alignments and get a recommendation of the model of sequence evolution to use for their analysis, as well as which alpha parameter to use (if any) in case of rate heterogeneity: We know that the position of each nucleotide in a sequence can have a different impact on the rate of substitution because of different evolutionary forces. To account for this, we use rate heterogeneity which is modeled with a gamma distribution. Changing the α parameter from this distribution we can increase or decrease the rate of heterogeneity. Otherwise, assuming same probability of substitution on each nucleotide means having rate homogeneity.

Results show that ModelRevelator performs on par with maximum likelihood with Bayesian information criterion, being able to generalize to alignments quite different from the ones that were trained on, as well as being more computationally inexpensive for certain alignment sizes and even outperforming in some tested data sets.

2 Objectives

The objective and goal of this project is to improve the preprocessing of multiple sequence alignments given to NNmodelfind by computing summary statistics on subsets of the multiple sequence alignments, trying to maximize the information extracted from them, and therefore the performance of NNmodelfind.

3 Methods

3.1 Data simulation

ModelRevelator was trained using alignments that were simulated using model parameters that were extracted from empirical data, collected from the scientific literature by Rob Lanfear from the Lanfear group of the Australian National University, with the objective of using empirical data for the simulation. The trees for alignment simulation were random, with internal and external branch lengths recovered from trees reconstructed on the empirical Lanfear data under the GTR model. The branch lengths were then used to generate the trees with 8, 16, 64 or 128 taxa.

In this project, we created two training datasets following the above steps, one with 86020 alignments and the other with 240000 alignments per file for each of the five preprocessing strategies that will be explained next. For each training dataset we created a respective validation dataset, of 3000 alignments and 12000 each.

To create the test dataset, we followed the same strategy as for the training dataset, but used different sequence lengths to see if the models could generalize to other sequence lengths. We created alignments with 100, 1000, 10000 and 100000 base pairs.

3.2 Preprocessing strategies

We used the Seq-Gen[18] tool to use these created trees in combination with some parameters (rate frequency, sequence length and alpha) for each model of substitution to simulate a multiple sequence alignment, using the 12 different models of sequence evolution in equal parts: half with rate homogeneity and half with rate heterogeneity across sites. The rate heterogeneity was set to alphas of 0.001, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10. All the sequences from the multiple sequence alignment in the training dataset are 1000bp long.

The first step for improving the preprocessing of the multiple sequence alignments was to choose the different approaches to use. ModelRevelator uses a pairwise strategy (as explained below) on the preprocessing strategy since it yielded the best results, but several other methods were tested (also explained below). However, these strategies might be further optimized since extracting the necessary information can probably be done without the need of looking at each sequence entirely, or even only looking at certain sequences of interest. Since we don't know how the information will be best extracted from the alignments, we need to look at the already existing methods that were proven to work in ModelRevelator:

- **Column-wise:** For each column of the multiple sequence alignment the following features were counted: πA, πC, πG, πT, AC, AG, AT, CT, CG, GT, AA, CC, GG, TT.

- **Pair-wise:** For two randomly selected sequences of the multiple sequence alignments the following features were computed: AA, CC, GG, TT, first strand base count A first, strand base count C, first strand base count G, first strand, base count T, second strand base count A, second strand, base count C, second strand base count G, second strand, base count T, transversion count, transition count, AC, AG, AT, CT, CG, GT, CA, GA, TA, TC, GC, TG.

The next step was to apply these strategies to each individual subset and think of the best ways to do so. All of this resulted in 5 different strategies used to try to extract the maximum amount of information from the alignments to give to the neural network:

I **Column-wise subsets**: The first strategy consists of using the column-wise strategy explained before to each of the subsets and computing the previously mentioned summary statistics for all of the columns. Then all of the summary statistics computed get concatenated and reshaped using Numpy to get the 200 by 200 by 16 tensor that we will use as input for the neural network. We can see a representation of this strategy in figure 2

II **Pairwise subsets**: Implementation of the pairwise strategy to each subset and concatenating the summary statistics to get a 200 by 200 by 26 tensor as done in I. We can see a representation on this strategy in figure 3

III **Column-wise random sized subsets**: In this strategy we are doing the same as the first strategy of column-wise subsets, but we are randomly selecting the size of each subset from a range of 80 to 160, and then only computing and storing the first 20 summary statistics to the tensor, which has size 100 by 100 by 16.

IV **Pairwise random sized subsets**: Same as in 3, but applied to the pairwise strategy. We end up with a tensor of 100 by 100 by 26.

V **Choosing the more complex sequences**: We theorized that the best way for the neural network to extract information and patterns from the summary statistics computed is to use the more complex sequences, meaning the ones that have the most differences from the rest. Therefore, after using the same strategy as the previous pairwise random sized subsets, we computed the summary statistics for all the random-sized subset, and picked the 20 most complex sequences to be stored on the 100 by 100 by 26 tensor. We can see a representation on this strategy in figure 4

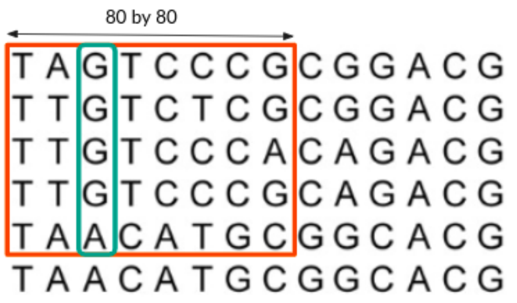

80 by 80
T A G T C C C G C G G A C G
T T G T C T C G C G G A C G
T T G T C C C A C A G A C G
T T G T C C C G C A G A C G
T A A C A T G C G G C A C G
T A A C A T G C G G C A C G

Figure 2: *Column wise strategy. In orange we have the subset, and in green the selected column*

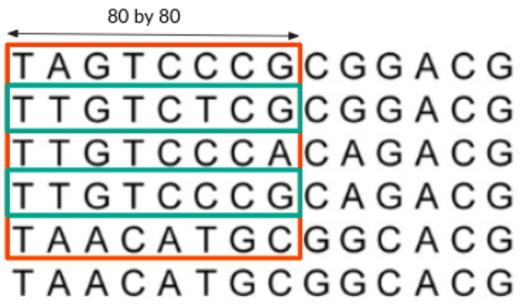

80 by 80
T A G T C C C G C G G A C G
T T G T C T C G C G G A C G
T T G T C C C A C A G A C G
T T G T C C C G C A G A C G
T A A C A T G C G G C A C G
T A A C A T G C G G C A C G

Figure 3: *Pairwise strategy. In orange we have the subset, and in green the selected rows*

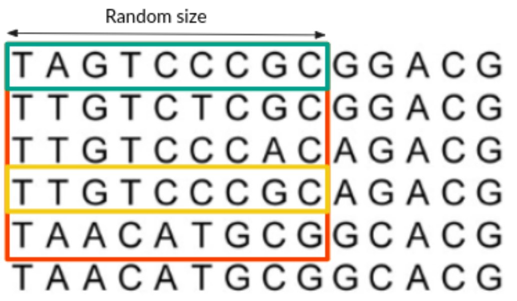

Random size
T A G T C C C G C G G A C G
T T G T C T C G C G G A C G
T T G T C C C A C A G A C G
T T G T C C C G C A G A C G
T A A C A T G C G G C A C G
T A A C A T G C G G C A C G

Figure 4: *Complex sequences. In orange we have the subset, in green the possible columns and in yellow the chosen more complex column*

In all methods the summary statistics are normalized by the sequence length of the multiple sequence alignment, in this case 1000bp. Each subset has size of 80 by 80 by default, and considering we are only using four sizes of taxa (8, 16, 64, 128) and 12 possible models of evolution, we reshaped the tensor accordingly. For example the first strategy has a 200 by 200 by 16 tensor because we are computing 80 summary statistics, and 16 is the number of features we are computing. But then for the column-wise with random size subsets we have 100 by 100 by 16 because instead of 80 summary statistics we are only using the first 20 in the tensor.

To apply this methods to the subsets we started implementing the column wise approach to the subsets: For a random position of the multiple sequence alignment the subset of size 80 was created, and the summary statistics from the column wise method was applied to this subset in addition to transition and transversion count and position of the subset. This was done 500 times, and at the end a Numpy reshape function was applied to end up with a 200 by 200 by 16 (number of features computed) tensor, stored in tfrecord format, the TensorFlow[5] binary data storage format.

The first strategy returns a 200 by 200 by 16 tensor because we are computing 80 summary statistics for each of the 500 subsets (80 * 500 = 200 * 200), and 16 is the number of features we are computing. But then for the column-wise with random sized subsets we have 100 by 100 by 16 because instead of 80 summary statistics we are only using the first 20 in the tensor and we have fewer features.

3.3 Neural network architecture

This section will aim to describe the neural network architecture of NNmodelfind which was used for this project. To do so, we first need to describe what convolutional and residual neural networks are.

3.3.1 Convolutional neural networks

There is a specialized type of NN called convolutional neural networks (CNN) that works best for two-dimensional dependency of the data points, like the pixels in an image. These networks are given this name because they use convolution (a mathematical linear operation in which a function is modified by another) in at least one layer.
In this type of network (Fig6), each layer is responsible for finding different patterns in increasing complexity on the inputs. There are three steps happening on each of these convolutional layers: In the first one we perform some convolutions in parallel, using something called a filter, which is usually a matrix learned during training used, for example, to detect edges of a picture. They are also frequently referred to as channels. We use the result of this convolution to produce a non linear activation function like a rectified linear unit[25]. In the second step, the results are then given to a nonlinear activation function such as the rectified linear function (which is the one we use, and only takes the positive part of the data). The last step is pooling, which consists of summarizing the information given to the next layer to reduce the computational time without losing much information.

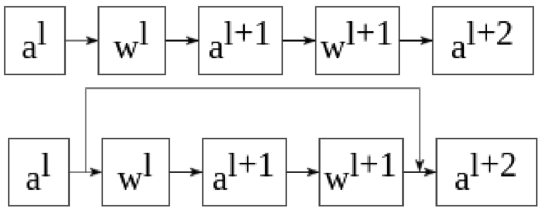
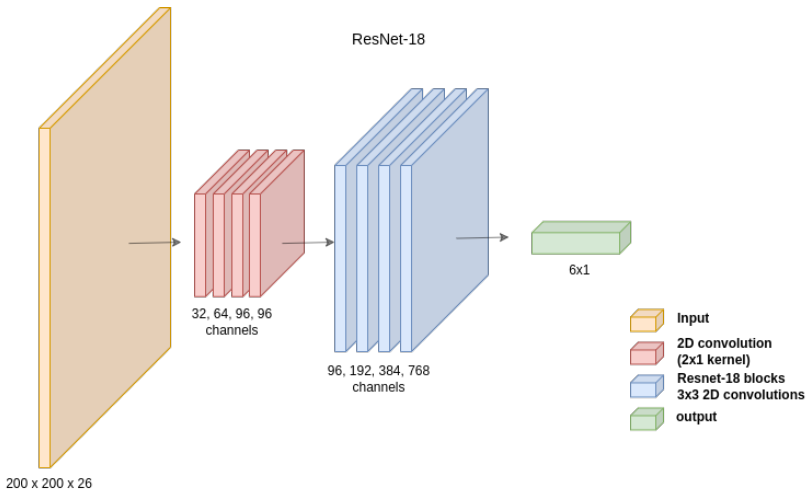


Figure 6: *Diagram of two layers of a normal CNN and two layers of a residual block, highlighting the skip connection. a = activation function, w = weight layer.*

3.3.2 Residual neural networks

One of the issues with CNNs is that for very deep neural networks there is a saturation in the accuracy of these deeper layers (vanishing gradient error[2]), as well as a higher training error.
To solve this issue, we will be using a residual neural network or ResNet[8], which also aims to improve the training of deep neural networks by reducing the problem of vanishing gradient, in which the gradient is so small that the weight barely changes, hindering future training improvement.
ResNets do this by implementing residual blocks (Fig6), which have the same structure as a convolutional layer, but the output for an activation function on one layer will be later added before the nonlinear activation function n layers down, skipping n layers in the middle.



ResNet-18
32, 64, 96, 96 channels
96, 192, 384, 768 channels
6x1
200 x 200 x 26
Input
2D convolution (2x1 kernel)
Resnet-18 blocks 3x3 2D convolutions
output

Figure 5: *NNmodelfind architecture*

This is called "skip connection" when at least one layer is skipped. NNmodelfind uses a 1×1 convolutional shortcut as seen in [8].

ModelRevelator uses a ResNet-18 (Fig5), starting with a tensor input of varying size depending on the strategy used. This input is passed to four 2D convolution layers with a 2x1 kernel that have 32, 64, 92 and 92 channels each instead of the original ResNet-18 7x7 convolution. After pooling, the output is then passed to the 4 ResNet-18 blocks that use 3x3 convolutions and 96, 192, 384 and 768 channels respectively. From that we get a 6x1 layer of the output, out of the 6 possible models of sequence evolution (notice that we use 6 and not 12 because we separate the alpha estimation and the model selection into two different neural networks). The network uses categorical cross-entropy as the loss-function, and Adam optimizer[13] for the gradient descent.

3.4 Training and testing

After creating the first training dataset of 86020 alignments and the validation dataset of 3000 we trained with a validation dataset 2 of the 5 different proposed strategies: Column wise summary statistics and pairwise summary statistics both with random subsets sizes. The training was done using TensorFlow 2.8 version on the following GPUs: Nvidia Tesla V100-PCIE with 32Gb of memory or Nvidia GeForce RTX2080Ti with 11Gb of memory. Both trainings took around 10 days.

For all the training we used the following hyperparameters: learning rate of 10^{-5} and batch size of 40. We performed early stopping on both trainings to aim for convergence (finding the minimum for the loss function). This happened on epoch 100 for the pairwise strategy and epoch 60 on the column wise strategy as we can see in Figures 7 and 8, on which TensorBoard[5] showcases both trainings.

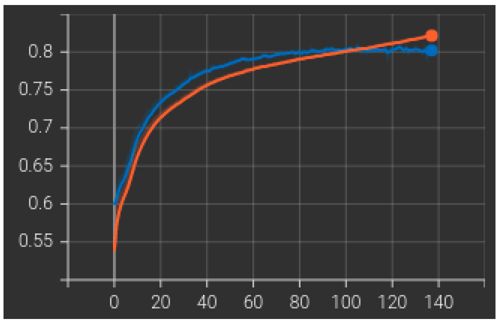


Figure 7: *Representation of the pairwise strategy training using Tensorboard. In orange we have the training accuracy and in blue the validation accuracy. x-axis = epochs, y-axis = accuracy*

To test the models, we used the simulated test dataset for each individual alignment length and tested each of the two best performing models.

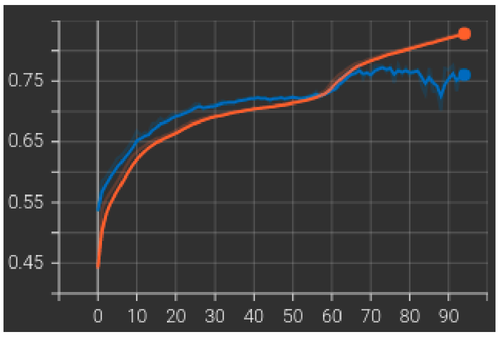


Figure 8: *Representation of the column wise strategy training using Tensorboard.*

4 Results

Table 1: *Training results. T = Training V = Validation, A = accuracy, L = loss.*

Strategy	Epoch	TA	TL	VA	VL
Pairwise	106	80.42	1.448	80.92	1.438
Column wise	67	78.07	1.53	77.44	1.556

The results from training the two different strategies can be seen in Table 1 and in Figure 8. For the column wise strategy, we can see some overfitting after 60 epochs with an accuracy of around 75% for both the validation dataset and the training dataset, while the pairwise approach took around 100 epochs for convergence and had an accuracy of 80% on the training and validation datasets. As we can see the pairwise strategy took longer to converge (40 more epochs) and yielded better results (5%) than the column wise strategy, which is expected as will be discussed in the next section.

To check the test results, we will compare both previously mentioned trained strategies to the results from ModelRevelator as well as the traditional method of Maximum Likelihood. Figures 11 and 9 show the accuracy of the pairwise summary statistics data with random subset sizes compared to the column wise summary statistics with random subset sizes for the 1000bp lengths in the test dataset and for each of the 6 models of evolution in increasing order of complexity.

Figures 9 and 11 show the results from the training in this project, and figures 10 and 12 show original data from ModelRevelator, comparing the training that was done to Maximum Likelihood in the same fashion. Figures 13 and 15 shows the pairwise and column wise results for the 100000bp test dataset and figures 14 and 16 the same thing for ModelRevelator and Maximum Likelihood. Finally figures 17 and 19 show the training on the 100bp test dataset, while 18 and 20 show the original data tested on the 100bp test dataset on ModelRevelator.

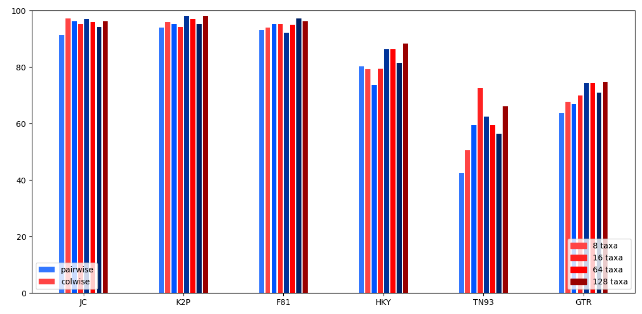


Figure 9: *Model selection performance on 1000bp rate homogeneus test dataset for pairwise and column wise approaches, using taxas colored respectively in decreasing color intensity for each of the 6 models in increasing order of complexity*

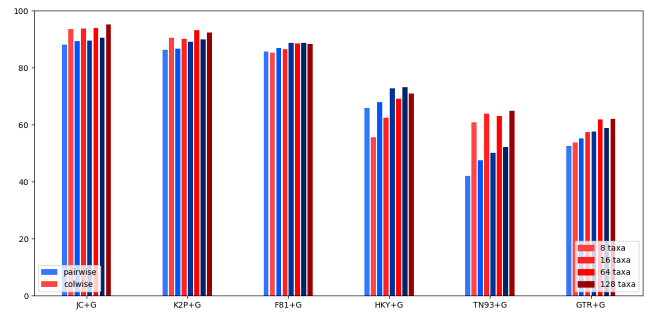


Figure 11: *Model selection performance on 1000bp rate heterogeneous test dataset for pairwise and column wise approaches.*

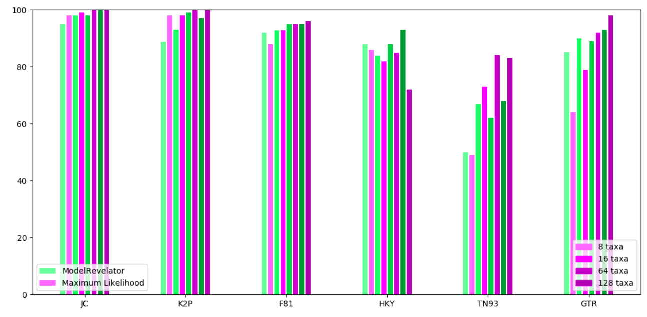


Figure 10: *Model selection performance on 1000bp rate homogeneous test dataset from the original data of ModelRevelator and Maximum Likelihood.*

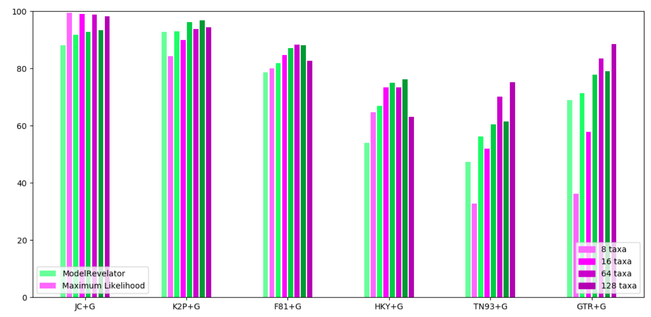


Figure 12: *Model selection performance on 1000bp rate heterogeneous test dataset from the original data of ModelRevelator and Maximum Likelihood.*

As we can see, the results from the preprocessing of the subsets are at most 7% lower than the ones used in ModelRevelator and Maximum Likelihood on all the models but 20% lower for GTR (the most complex model) on the 1000bp test dataset. This pattern is also seen on the largest sequences in the test datasets: The models are able to extract more information for longer sequences in 100000bp test datasets, while they struggle with more complex models.

For shorter sequences as seen for 100bp, the models cannot extract enough information, so the results are quite different: The accuracy is lower than 50% on all models except for HKY and GTR, which are more complex.

This phenomenon was also observed in ModelRevelator, and it is the opposite of what we see with Maximum Likelihood, and with our approach on longer sequences, where the more complex (parameter-rich) the model of sequence evolution is, the lower the accuracy is achieved. We can also see that the column wise approach works substantially better than the pairwise strategy, having 10 to 20 percent higher accuracy in all models and taxa except for GTR, where they perform more similarly.

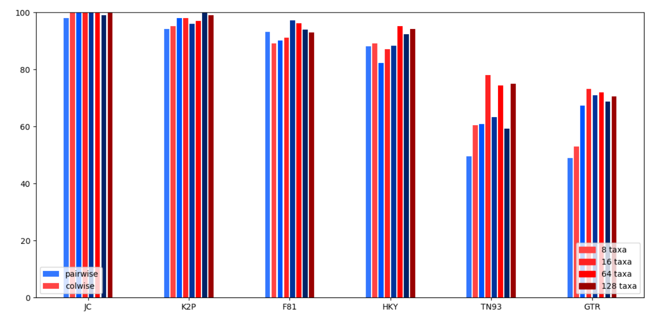


Figure 13: *Model selection performance on 100000bp rate homogeneous test dataset for pairwise and column wise approaches.*

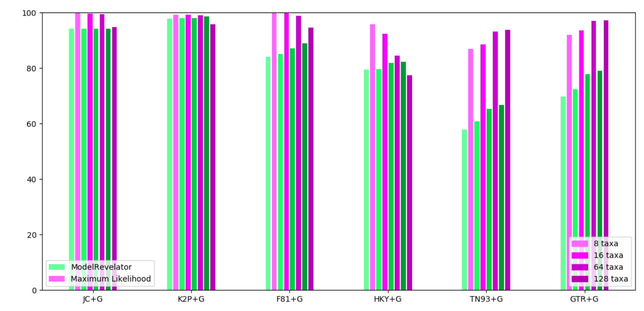


Figure 14: *Model selection performance on 100000bp rate homogeneous test dataset from the original data of ModelRevelator and Maximum Likelihood.*

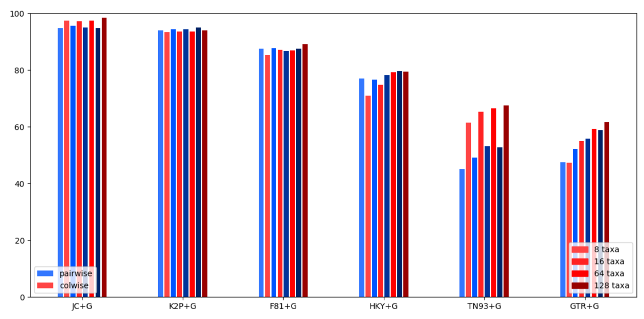


Figure 15: *Model selection performance on 100000bp rate heterogeneous test dataset for pairwise and column wise approaches.*

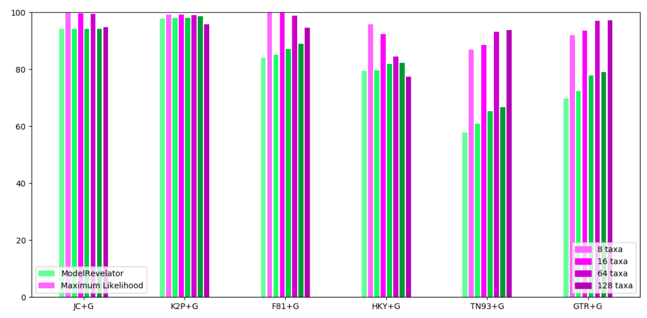


Figure 16: *Model selection performance on 100000bp rate heterogeneous test dataset from the original data of ModelRevelator and Maximum Likelihood.*

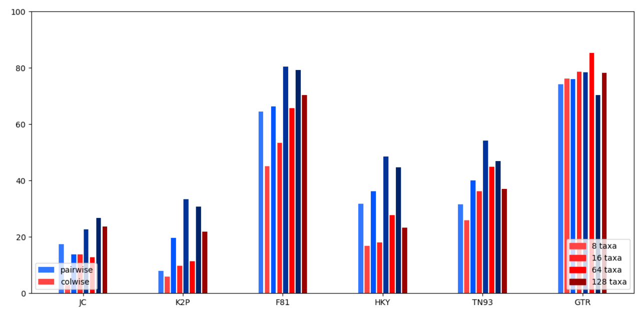


Figure 17: *Model selection performance on 100bp rate homogeneous test dataset for pairwise and column wise approaches.*

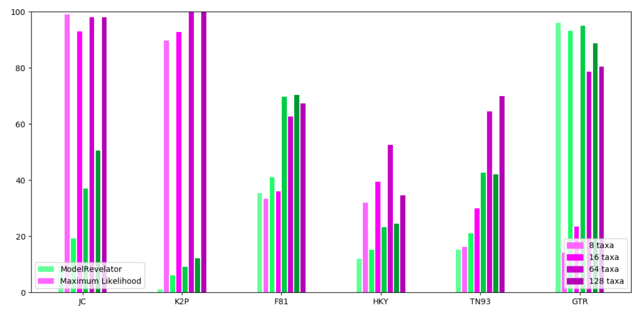


Figure 18: *Model selection performance on 100bp rate homogeneous test dataset from the original data of ModelRevelator and Maximum Likelihood.*

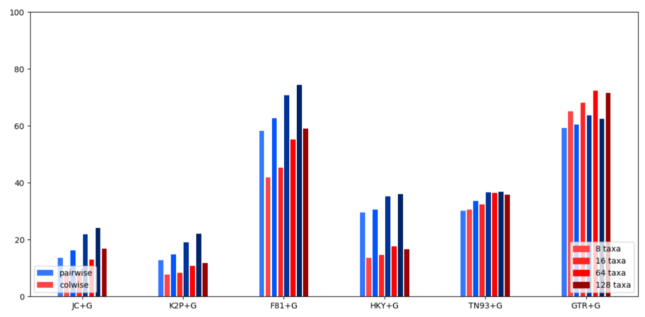


Figure 19: *Model selection performance on 100bp rate homogeneous test dataset for pairwise and column wise approaches.*

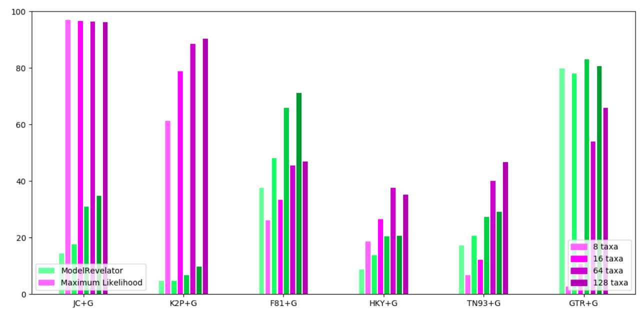


Figure 20: *Model selection performance on 100bp rate heterogeneous test dataset for pairwise and column wise approaches.*

When comparing the performance of the models on alignments with or without rate heterogeneity, we can see that the accuracy of the models when tested on data with rate homogeneous is around 4% better on all the models for the 1000bp test dataset, especially for TN93 and GTR on larger taxa, where it increases to 11% on average for taxa 16 64 and 128. It is interesting to point out that in all the results except for the 100bp test dataset, the performance of both strategies is quite similar, except for TN93, where the column wise approach outperforms the pairwise by at least 10 to 20% for 16, 64 and 128 taxa.

5 Discussion

The performance of the new strategies is similar to the ones previous obtained in ModelRevelator. Looking at the results we can say that it is almost on par with the results obtained by the previous methods in NNmodelfind on the simpler models. However, the performance is not as good on the more complex models, probably because more complex parameters are harder to summarize for the neural network to learn. From that we can hypothesize that only using the information contained in a subset might not be enough for the neural network to learn all the patterns with the strategies trained so far. Thus, it seems as if computing the summary statistics from subsets makes it easier to estimate those models of sequence evolution that have fewer parameters, but this type of input seems to be less efficient for the neural network when looking at more parameter-rich models, like GTR.

To solve this issue, the next step for increasing the accuracy is to test the other strategies mentioned in the methods section that will be used downstream on the project, and see if the performance on the more complex models improves.

When looking at the two strategies of column wise and pairwise summary statistics on random sized subsets that were trained, we see that, even though the pairwise strategy had more features and showed higher accuracy on the training and validation datasets, the results do not indicate whether one is better than the other. This is surprising, since the pairwise strategy performed better than the column wise on ModelRevelator when looking at the whole sequence, although shorter pairwise summary statistics certainly contain less information than the whole sequence, so retrieving information from the alignment might be strongly dependent on the sequence length.

For some reason, the column wise approach gives higher accuracy on TN93, probably because column-wise summary statistics inputs always provide information of both transition types in each summary statistic of each position of the alignment. This is not the case for the pairwise input, as the information is collected over the whole alignment, not just for each position, making it harder for the neural network to differentiate between the two transitions types that TN93 has as parameters.

However, looking at the 100bp test dataset we can see that the pairwise strategy performs around 20% better in all models except GTR than the column wise approach, which might mean that the pairwise strategy generalizes slightly better, at least for shorter sequences.

Another thing we can observe from the results, is that the performance of both models is not as substantially high as we would expect on the 100000 base pairs test dataset as compared to the length of 1000 base pairs with which we trained. This is probably explained by the fact that for both strategies the same number of subsets were looked at, independently of the test dataset sequences length.

Overall, the results are very similar to the ones obtained with the previous preprocessing strategies in ModelRevelator. However, the neural network has a harder time selecting the correct model if this is quite a complex one. It would be very interesting to compare these results with other known machine learning methods for model of substitution estimation, but the closest thing published is ModellTeller[1], which uses a random forest algorithm to predict models of substitution for branch-length estimation. This process requires tree estimation, making the process different enough so that the comparison is beyond the scope of the results obtained now. Nevertheless, it might be interesting to do so in the future.

6 Conclusion

In this project, we tried to create a new preprocessing strategy for multiple sequence alignments to be used by a neural network for estimating the model of sequence evolution. We wanted this strategy to improve in both performance and time savings, compared to established methods and the preprocessing approach used in ModelRevelator. The results obtained show that, even though the new strategies' performance is quite similar to those of ModelRevelator and Maximum Likelihood for the simpler models, the created strategies struggle to accurately select the more complex models. Further work on the project will be needed to test the aforementioned strategies and see if the results improve.

To conclude this project, we hope that the results achieved showcase how powerful a tool machine learning can be in the bioinformatics field. Further resources should be applied to use these tools on phylogeny and other relevant health science fields to solve or optimizing the solution to pressing biological problem.

7 Data availability

All the codes and programs used to carry out this project can be found in github: `https://github.com/antonvegam/ModelRevelator_subsets`

8 Acknowledgements

I would like to express my sincerest gratitude to all of the CIBIV group for having me, to Arndt von Haeseler for giving me the opportunity to work there, and specially to Sebastian Burgstaller-Muehlbacher for guiding me through this project and teaching me all I needed to know and more.

Bibliography

[1] Shiran Abadi et al. "Modelteller: Model selection for optimal phylogenetic reconstruction using machine learning". In: *Molecular Biology and Evolution* 37.11 (2020), pp. 3338–3352. ISSN: 15371719. DOI: 10.1093/molbev/msaa154. URL: https://academic.oup.com/mbe/article/37/11/3338/5862639.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. ISSN: 19410093. DOI: 10.1109/72.279181.

[3] Sebastian Burgstaller-Muehlbacher et al. "ModelRevelator: Fast phylogenetic model estimation via deep learning". In: *bioRxiv* (2021).

[4] Joseph Felsenstein. “Evolutionary trees from DNA sequences: A maximum likelihood approach”. In: *Journal of Molecular Evolution 1981 17:6* 17.6 (1981), pp. 368–376. ISSN: 1432-1432. DOI: 10.1007/BF01734359. URL: https://link.springer.com/article/10.1007/BF01734359.

[5] Google Brain. *TensorFlow: A System for Large-Scale Machine Learning | USENIX*. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi (visited on 03/22/2022).

[6] Peter J. Green. “Reversible jump Markov chain Monte Carlo computation and Bayesian model determination”. In: *Biometrika* 82.4 (1995), pp. 711–732. ISSN: 0006-3444. DOI: 10.1093/BIOMET/82.4.711. URL: https://academic.oup.com/biomet/article/82/4/711/252058.

[7] Masami Hasegawa, Hirohisa Kishino, and Taka aki Yano. “Dating of the human-ape splitting by a molecular clock of mitochondrial DNA”. In: *Journal of molecular evolution* 22.2 (1985), pp. 160–174. ISSN: 0022-2844. DOI: 10.1007/BF02101694. URL: https://pubmed.ncbi.nlm.nih.gov/3934395/.

[8] Kaiming He et al. “Identity Mappings in Deep Residual Networks”. In: (). arXiv: 1603.05027v2. URL: https://github.com/KaimingHe/.

[9] THOMAS H JUKES and CHARLES R CANTOR. “CHAPTER 24 - Evolution of Protein Molecules”. In: *Mammalian Protein Metabolism*. Ed. by H N MUNRO. Academic Press, 1969, pp. 21–132. ISBN: 978-1-4832-3211-9. DOI: https://doi.org/10.1016/B978-1-4832-3211-9.50009-7. URL: https://www.sciencedirect.com/science/article/pii/B9781483232119500097.

[10] Subha Kalyaanamoorthy et al. “ModelFinder: fast model selection for accurate phylogenetic estimates”. In: *Nature Methods* 14.6 (2017), pp. 587–589. ISSN: 1548-7105. DOI: 10.1038/nmeth.4285. URL: https://doi.org/10.1038/nmeth.4285.

[11] Gaurav Kandoi, Marcio L. Acencio, and Ney Lemke. “Prediction of druggable proteins using machine learning and systems biology: A mini-review”. In: *Frontiers in Physiology* 6.DEC (2015), p. 366. ISSN: 1664042X. DOI: 10.3389/FPHYS.2015.00366/BIBTEX.

[12] Motoo Kimura. “A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences”. In: *Journal of molecular evolution* 16.2 (1980), pp. 111–120. ISSN: 0022-2844. DOI: 10.1007/BF01731581. URL: https://pubmed.ncbi.nlm.nih.gov/7463489/.

[13] Diederik P. Kingma and Jimmy Lei Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2014). DOI: 10.48550/arxiv.1412.6980. arXiv: 1412.6980. URL: https://arxiv.org/abs/1412.6980v9.

[14] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature 2015 521:7553* 521.7553 (2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://www.nature.com/articles/nature14539.

[15] Agnieszka Mikołajczyk and Michał Grochowski. “Data augmentation for improving deep learning in image classification problem”. In: *2018 International Interdisciplinary PhD Workshop, IIPhDW 2018* (2018), pp. 117–122. DOI: 10.1109/IIPHDW.2018.8388338.

[16] Marco Salemi Philippe Lemey and Anne-Mieke Vandamme. *The Phylogenetic Handbook*. Cambridge. ISBN: 9780521877107.

[17] David Posada and Thomas R. Buckley. “Model Selection and Model Averaging in Phylogenetics: Advantages of Akaike Information Criterion and Bayesian Approaches Over Likelihood Ratio Tests”. In: *Systematic Biology* 53.5 (2004), pp. 793–808. ISSN: 1063-5157. DOI: 10.1080/10635150490522304. URL: https://academic.oup.com/sysbio/article/53/5/793/2842928.

[18] Andrew Rambaut and Nicholas C Grassly. “Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees”. In: *Bioinformatics* 13.3 (1997), pp. 235–238.

[19] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature 1986 323:6088* 323.6088 (1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0.

[20] Charumathi Sabanayagam et al. “A deep learning algorithm to detect chronic kidney disease from retinal photographs in community-based populations”. In: *The Lancet Digital Health* 2.6 (2020), e295–e302. ISSN: 25897500. DOI: 10.1016/S2589-7500(20)30063-7/ATTACHMENT/5DD9F8AF-9E3D-4DCC-9CE8-EBDA8B19764A/MMC1.PDF. URL: http://www.thelancet.com/article/S2589750020300637/fulltexthttp://www.thelancet.com/article/S2589750020300637/abstracthttps://www.thelancet.com/journals/landig/article/PIIS2589-7500(20)30063-7/abstract.

[21] Katharina Schwarze et al. “The complete costs of genome sequencing: a microcosting study in cancer and rare diseases from a single center in the United Kingdom”. In: *Genetics in Medicine 2019 22:1* 22.1 (2019), pp. 85–94. ISSN: 1530-0366. DOI: `10.1038/s41436-019-0618-7`. URL: `https://www.nature.com/articles/s41436-019-0618-7`.

[22] K. Tamura and M. Nei. “Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees”. In: *Molecular biology and evolution* 10.3 (1993), pp. 512–526. ISSN: 0737-4038. DOI: `10.1093/OXFORDJOURNALS.MOLBEV.A040023`. URL: `https://pubmed.ncbi.nlm.nih.gov/8336541/`.

[23] Adi L. Tarca et al. “Machine Learning and Its Applications to Biology”. In: *PLOS Computational Biology* 3.6 (2007), e116. ISSN: 1553-7358. DOI: `10.1371/JOURNAL.PCBI.0030116`. URL: `https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.0030116`.

[24] S Tavaré. “Some probabilistic and statistical problems on the analysis of DNA sequences”. In: *Lectures on Mathematics in the Life Sciences* 17 (1986), pp. 57–86.

[25] Geoffrey E. Hinton Vinod Nair. *Rectified Linear Units Improve Restricted Boltzmann Machines | OpenReview*. URL: `https://openreview.net/forum?id=rkb15iZdZB` (visited on 06/20/2022).

[26] Ziheng Yang. “Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods”. In: *Journal of Molecular Evolution* 39.3 (1994), pp. 306–314. ISSN: 14321432. DOI: `10.1007/BF00160154`.