BMI Homework 3

Matt Jones

February 23, 2018

Contents

1 Introduction

In this assignment I implemented the Smith-Waterman algorithm for local alignments of sequences, and benchmarked its performance on various substitution matrices and parameterizations. BLOSUM50, BLOSUM62, MATIO, PAM100, and PAM250 were all tested while documenting the algorithms performance in aligning the sequences provided.

This document will outline the implementation of the Smith-Waterman algorithm, the benchmarks performed, and then finally an algorithm used to optimize a substitution matrix for the alignment of known positive and negative sequences.

2 The Smith-Waterman Algorithm

The Smith-Waterman algorithm implemented here utilizes a dynamic programming approach to align two sequences, s_1 and s_2 according to some substitution score matrix provided as input. Briefly, we utilize a dynamic programming table H which is instantiated to be an $(M+1)\times(N+1)$ matrix of -1's. Here M is the length of s_1 and N is the length of s_2. Notably, H must increase the length of dimensions in each direction by 1 to capture the base case where a sequence is aligned to an empty sequence – thus the first row and column all have scores of 0. Building up from here, we set $H_{i,j}$ according to the following rule:

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + S_{i,j} \\ max(H_{i-k,j} - W_k) \forall 1 \leq k < i \\ max(H_{i,j-m} - W_m) \forall 1 \leq m < j \\ 0 \end{cases}$$

In the recursive definition above, W_k and W_m are mismatch penalties whose definitions have been abstracted out for the simplicity of the equation above, but can take on different meanings. Generally, the these mismatch penalties can be defined as affine penalties with parameters for opening a gap, g, and extending an already existing gap, ν. Thus, $W_k = k\nu + g$. In the case of my algorithm, both g and ν can be modulated via command line, yet many implmentations assume a constant mismatch penalty where $\nu = 0$ and thus $W_{k/m} = g$ for all k or m.

2.1 Parameterizing the Affine Gap Penalty, W

As noted above, I've defined W as an affine penalty term for mismatches with two parameters g (the penalty of opening a gap) and ν (the penalty for continuing a gap). While this penalty schema offers the algorithm for flexibility in finding optimal alignments, there can be a decently sized search space to find the right values for g and ν.

For a first pass, I analyzed the False Positive Rate (FPR) as a function of different parameterizations. Specifically, I set the True Positive Rate to be 0.7 and found a score threshold where the proportion of scores for positively defined sequences was equal to 0.7. With this threshold, I found the FPR by reporting the proportion of negatively defined pairs of sequences whose scores exceeded this threshold. Importantly, the algorithm cannot identify "positively" and "negatively" defined pairs; rather these pairs are offered as input and thus surely play a role in the stability of the optimal parameters found.

For each pairing of g and ν, I report the FPR for the 50 negative pairs of sequences in Table 1. Ideally, we'd like to have as low a False Positive Rate as possible, so it seems that the combination of $g = 5$ and $\nu = 3$ offers the best parameterization with an FPR of 0.22.

Table 1: False Positive Rates for Combinations of g in 1-20 and ν in 1-5

g / ν	1	2	3	4	5
1	0.38	0.34	0.28	0.28	0.26
2	0.36	0.32	0.28	0.24	0.28
3	0.32	0.3	0.28	0.26	0.26
4	0.3	0.28	0.24	0.28	0.26
5	0.28	0.3	0.22	0.26	0.28
6	0.26	0.28	0.24	0.28	0.28
7	0.28	0.24	0.24	0.28	0.36
8	0.32	0.26	0.28	0.36	0.38
9	0.3	0.3	0.28	0.38	0.38
10	0.28	0.28	0.36	0.38	0.38
11	0.28	0.36	0.38	0.38	0.4
12	0.3	0.36	0.38	0.38	0.38
13	0.34	0.38	0.38	0.4	0.38
14	0.34	0.38	0.36	0.38	0.38
15	0.36	0.36	0.4	0.38	0.36
16	0.36	0.36	0.38	0.38	0.36
17	0.36	0.4	0.38	0.38	0.34
18	0.36	0.38	0.38	0.34	0.34
19	0.4	0.38	0.36	0.34	0.34
20	0.38	0.36	0.36	0.34	0.34

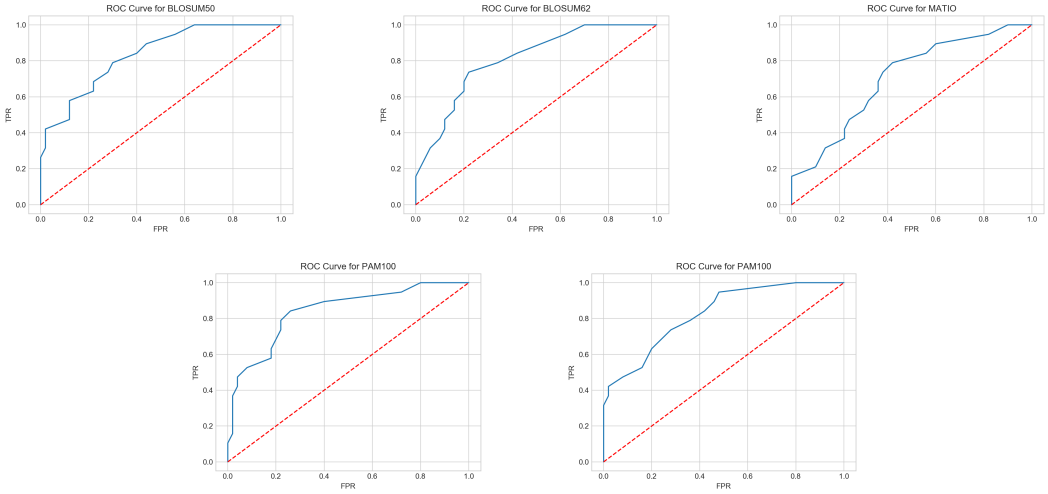
2.2 Identifying the Best Scoring Matrix

After identifying the optimal combination of parameters for my affine penalty function, I evaluated the impact of substitution score matrices on the algorithm's accuracy. To do so, I calculated ROC curves for each of the possible substitution score matrices - BLOSUM50, BLOSUM62, MATIO, PAM100, and PAM250, which are presented in Figure 1. To identify the best performing matrix, I minimized the FPR at a TPR of 0.7 as when identifying the optimal set of parameters. I found the following results:

- BLOSUM50 FPR = 0.22
- BLOSUM62 FPR = 0.21
- MATIO FPR = 0.37
- PAM100 FPR = 0.21
- PAM250 FPR = 0.26

It appears that both BLOSUM62 and PAM100 perform similarly when the true positive rate is set to be 0.7, so I'll select PAM100 arbitrarily to reference for downstream analyses.

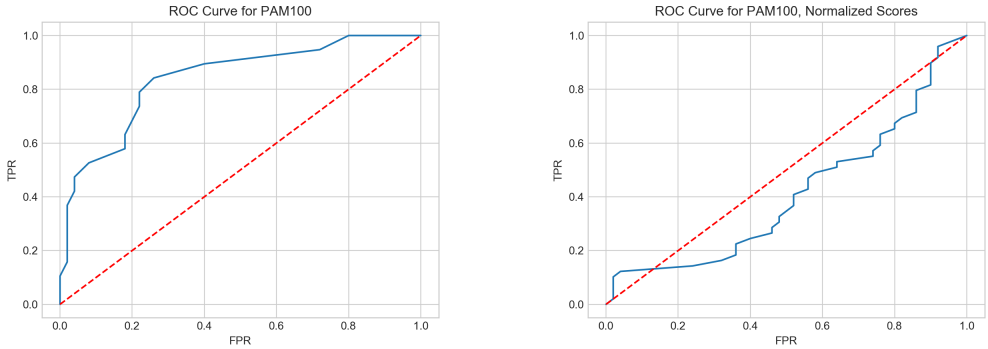
Figure 1: ROC Curves for Various Substitution Matrices



2.3 Normalizing Scores by Sequence Length

To address the dependence of the algorithm on the length of sequences, I compared ROC curves between scores where values in the matrix were normalized to the length of the shorter sequence and those where the matrix was not normalized. The difference is drastic, resulting in the ROC curve to fall significantly beneath the 1:1 reference line. The massive shift in the AUC may be intuitive, however – by normalizing the score of an alignment by the length of the shorter sequence, thresholds for true positive and false positive rates become more difficult to obtain. Looked at another way , but reducing the domain of scores for both positive and negative alignments, the transformed scores look much more similar to one another. Extracting from this trend, we'd expect that perhaps multiplying by a scalar value, we'd see that the distributions would be *less* similar, thus creating a more refined ROC curve. This analysis could be done simply, but will left for further work.

Figure 2: Comparisons of ROC Curves for Normalized Scores



3 Optimizing the Score Matrix

To further assess the ability of the score matrix to impact alignments and their scores, I developed a simple algorithm to optimize a given substitution matrix. In particular, I used a Monte Carlo - Markov Chain approach to stochastically change values in the matrix proportional to some

objective function. This seemed to be a logical application of an MCMC algorithm, and I was able to obtain drastically different matrices, which seem to perform better on the provided sequences.

3.1 The Algorithm

I follow a simple implementation of the MCMC algorithm, utilizing the following objective function:

$$S^* = argmax_S \sum_{FPR(\theta) \in [0.0, 0.1, 0.2, 0.3]} TPR(\theta)$$

Essentially, I try to find the substitution matrix S^* which maximizes the sum of true positive rates on the provided sequences for false positive rates in a given domain. Since we use the score matrix S to find the alignments scores for both negative and positive pairs, I can iteratively improve this function with my MCMC algorithm.

Before explaining the algorithm, it's also important to note that sequences were aligned only once with the optimal substitution matrix found before - PAM100. From these alignments, I calculated alignment scores for a pair of sequences relative to a substitution matrix as follows: $X = a - p$, where $a = \sum_{i,j aligned} S(i,j)$ (i, j are characters in the aligned sequences) and p is the affine gap penalty described above. Thus for static alignments, I'm able to iteratively re-evaluate the objective function for every new score matrix.

Finally, I will explain the algorithm: for $n \in 1..N$,

- For all $S_{i,j}$ add a value from a normal distribution centered at 0 with standard deviation 1. Let this new matrix be called S'. Note that this substitution matrix must be kept symmetric.
- Calculate alignment scores relative to S'.
- Evaluate the objective function, let the new value be equal to J'
- Now, assign $S = S'$ according to p:

$$p = \begin{cases} 1, & J' \geq J \\ e^{(J-J')n^T}, & else \end{cases}$$

For my application, I use T to be some constant to simulate an annealing process, allowing more the algorithm to explore outside of potential local optima.

3.2 Optimizing the PAM100 Matrix

To optimize the PAM100 substitution score matrix, I ran my stochastic algorithm for 200 iterations, with $T = 1$. After applying my algorithm, the new matrix seemed to have a much wider range of substitution scores, and because I was adding values from a normal distribution, all values became real values (as opposed to the discretized substitution scores before). In Figure 3, I compare the ROC curve I obtained from the optimization to the curve from the original PAM100 substitution matrix. Unfortunately, the ROC curves seem to indicate that the optimized matrix was *not* as effective in improving alignments as hoped. Particularly, the optimized ROC curve is significantly more deflated along the length of the curve, indicating that the true positive rates in the non-optimized analysis are much better. There are many reasons why this may be, but a primary reason could be because of the parameters I chose for my optimization. Should I have ran the optimizer longer, or have searched for an optimal T parameter, the performance of the optimized PAM100 matrix may have improved.

3.3 Optimizing the MATIO Matrix

I performed a similar procedure to optimize the MATIO substitution matrix, but increased the T parameter to 2, to evaluate the effects of this constant on optimizing the matrix. Qualitatively, I find that the optimized matrix looks similar to the optimized PAM100 matrix - entries are now real-valued and the range is substantially greater. To analyze the performance, I report a comparison of the ROC curves in Figure 4. Interestingly, once again the ROC curves seem to indicate that the optimized substitution matrix is not as effective in assigning scores as for the non-optimized matrix.

Figure 3: Comparisons of ROC Curves After Optimizations on PAM100

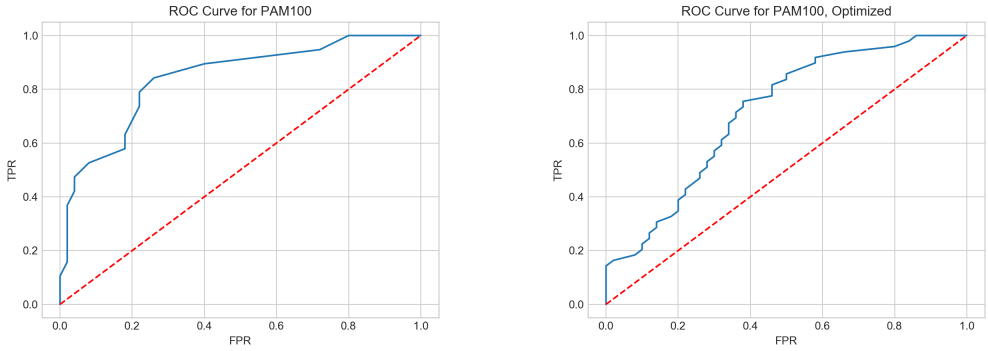
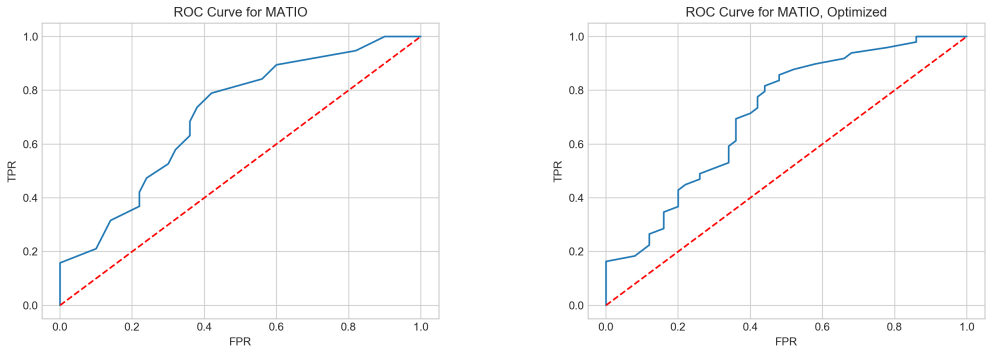


Figure 4: Comparisons of ROC Curves After Optimizations on MATIO



3.4 Discussion

I've introduced here a potential optimization algorithm for improving sequence alignment quality and scores. Although the algorithm does not seem to work perfectly, it is very possible that I just have not searched the full parameter space. There may be a certain set of parameters for every optimization problem - which could be unsettling, as this would seem to imply that the algorithm does not generalize well. Yet, many algorithms (e.g. Gradient Descent or Newton's Method) require certain hyperparameters (e.g. the learning rate) to work: this is merely a facet of optimization it seems, and is largely unavoidable. The algorithm itself could be improved in many ways - by changing the probability function, modulating T, or perhaps implementing novel ways to generate a new substitution score matrix.

To be sure, creating a powerful optimization algorithm and convincing people that it is powerful are two different things. While I've addressed the first question in the paragraph above, here I'll address the second. Mainly, I would need to show two things - that it increases precision and that it is generalizable. To show that it increases precision, ROC curves or Precision-Recall curves would suffice; especially when compared to matrices that BLAST or other local alignment tools use frequently. To show that it was generalizable, however, I would either need to perform iterative cross-validation (optimizing on a subset of data and performing on the held out data), or would have to acquire a novel set of data to test on. The latter of these two options is non-trivial – there may be many confounders when testing on a new dataset that was not acquired with the original dataset. Still, a great matrix should be able to perform well on this new dataset. Indeed, this is analogous to new sequences being added to some database: of course, we'd expect the new optimized matrix to perform well on these new sequences as well as the original sequences. Questions such as these will be left to further work.