# Smith Waterman Algorithm


## Use

`python main/run.py`

Uses fasta file paths specified in- * Pospairs.txt: true positive sequence alignments * Negpairs.txt: false positive sequence alignments

Can be tested with a smaller subset of pairs, each 10 sequences. * test_neg.txt * test_pos.txt

Auto-generates false positive rates, and ROC plots.

## Smith-Waterman implementation

### Works cited

Pseudocode consulted in Smith-Waterman implementation are as follows:

- http://vlab.amrita.edu/?sub=3&brch=274&sim=1433&cnt=1
- http://www.cs.kent.edu/~ssteinfa/papers/CASB_final.pdf
- https://gist.github.com/radaniba/11019717

### Picking best gap penalty and gap extension

Generated gap penalty combinations in the range of (1, 20) and gap extension penalties in the range (1, 5) for 50 positive sequence pairs and 50 negative sequence pairs. This was a total of 100 combinations of gap opening/extension penalties. False positive rates were calculated with a 0.7 true positive rate where false positives are the proportion of "negative" sequence pairs that pass the true positive threshold.

A sample of the results are in the following table:

| FPR | Gap Penalty | Gap Extension |
|---|---|---|
| 0.94 | 1 | 1 |
| 0.96 | 1 | 2 |
| 0.96 | 1 | 3 |
| 0.98 | 1 | 4 |
| 0.94 | 2 | 1 |
| 0.98 | 2 | 2 |

| FPR | Gap Penalty | Gap Extension |
|---|---|---|
| 0.96 | 2 | 3 |
| 0.94 | 2 | 4 |
| 0.94 | 3 | 1 |
| 0.98 | 3 | 2 |
| 0.98 | 3 | 3 |
| 0.9 | 3 | 4 |
| ... | ... | ... |
| 0.94 | 15 | 1 |
| 0.92 | 15 | 2 |
| 0.9 | 15 | 3 |
| 0.9 | 15 | 4 |

The full FPRs can be seen here.

The mean FPR for all 100 combinations is 0.934+/-0.0290. This is a relatively tight standard deviation, which indicates that the combinations of penalties have a slight, but noticeable affect on the algorithm.

Although overall trends were less apparent from summary statistics, looking at individual gap penalty/gap extension penalties revealed a strong preference for each.

The distribution of FPRs for gap extension showed preference for higher penalties.

This was continued with the gap opening penalties.

Interestingly, there seemed to be a limit in how large the gap extension penalty could be to see a yield in results for FPR reduction. A further direction would be to explore whether increasing the gap extension penalty has a similar tapering off affect.

Plotting in three-dimensional space revealed the best combination of penalties for the lowest FPR are (4, 17), (4, 18), (4, 19).

### Picking best scoring matrix

The best scoring matrix was picked out of five commonly used matrices, PAM250, PAM100, MATIO, BLOSUM62, and BLOSUM50.

For a range of true positive values, false positive rates were calculated using each of these matrices.

In these ROC curves, MATIO has by far the lowest true positive rates. PAM250, BLOSUM50 and BLOSUM62 all perform similarly. PAM100 seems to be slightly more sensitive than the the other matrices at some points.

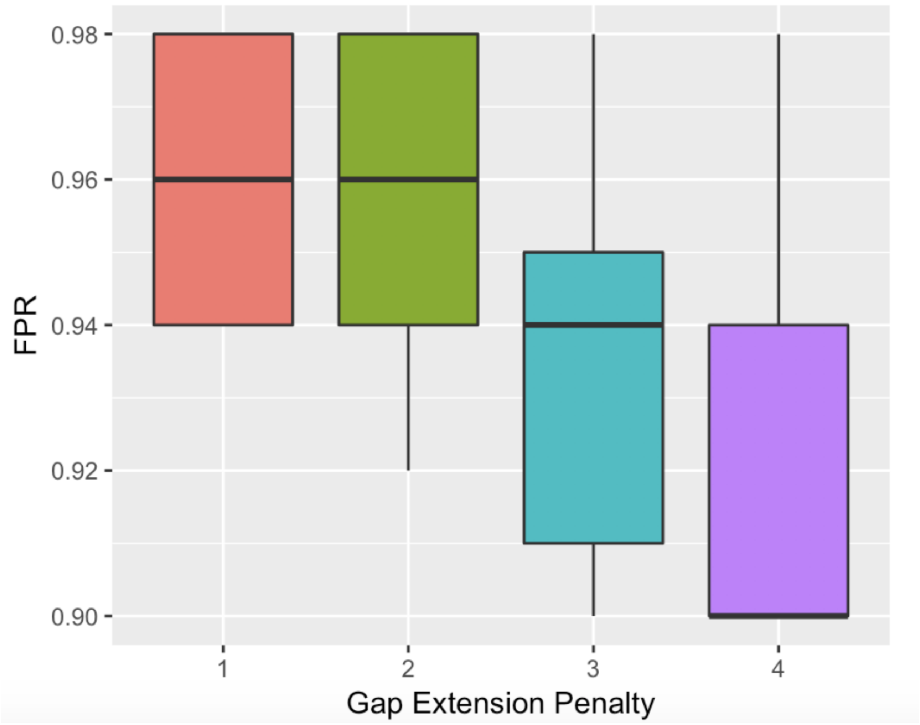


Figure 1: image1

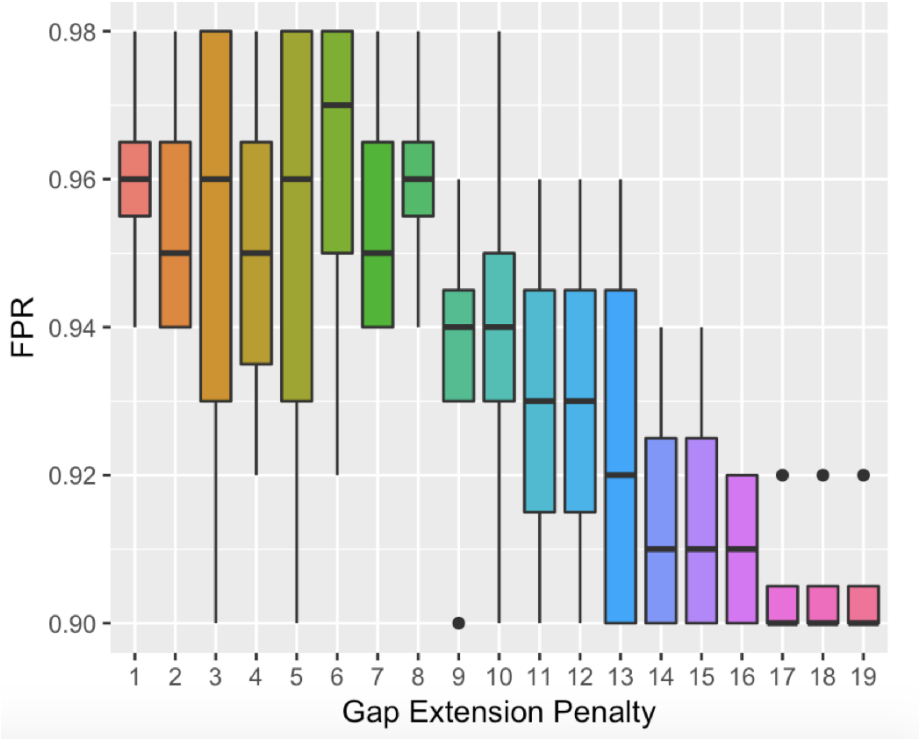


Figure 2: image2

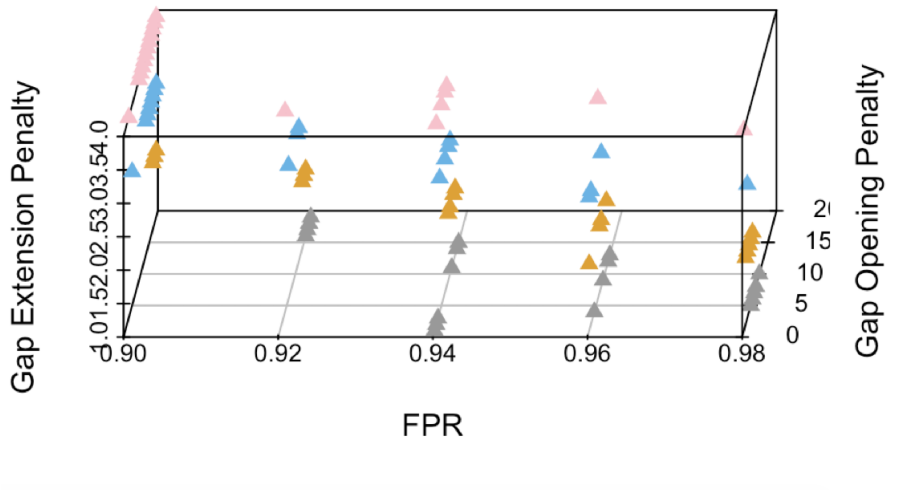


Figure 3: image3

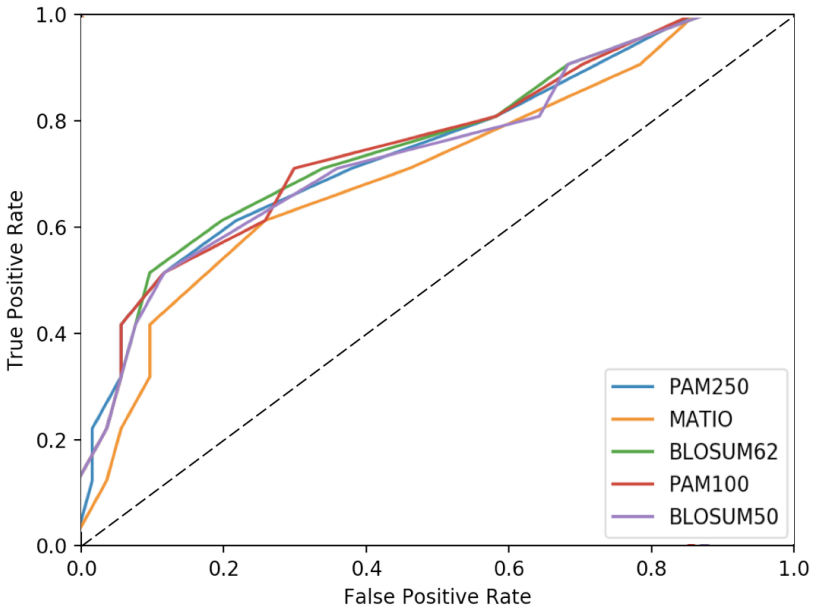


Figure 4: image4

In my code, I implemented picking the 'best' scoring matrix by picking the lowest average FPR. However, by this metric alone, I would have picked MATIO:

| Matrix | Average FPR |
|---|---|
| PAM250 | 0.723 |
| MATIO | 0.685 |
| BLOSUM62 | 0.730 |
| PAM100 | 0.729 |
| BLOSUM50 | 0.718 |

It is clear from the ROC plot that MATIO is the worst performing matrix in terms of *true positives*, thus we will have to pick a trade-off of sensitivity and specificity for our scoring matrix. I believe that based on this data, BLOSUM62 performs best.

### Normalization

Given that the lengths of sequences in each of the positive and negative pairs varies substantially, normalizing by sequence length is a popular technique to prevent bias. For example in the smith-waterman algorithm the sequences

```
seq1: ATACAGAGAGTTTCACAGATACTACGGGA

seq2: ----------TTTCACA----------
```

Would have large gap penalties attached to their alignment. This may be unfair, because in reality, the alignment is quite good.

Using the BLOSUM62 matrix and penalties of 17, 4, I tried a simple normalization. I found that this did improve my sensitivity/specificity trade off, suggesting that in the future, normalization could help yield less biased results.

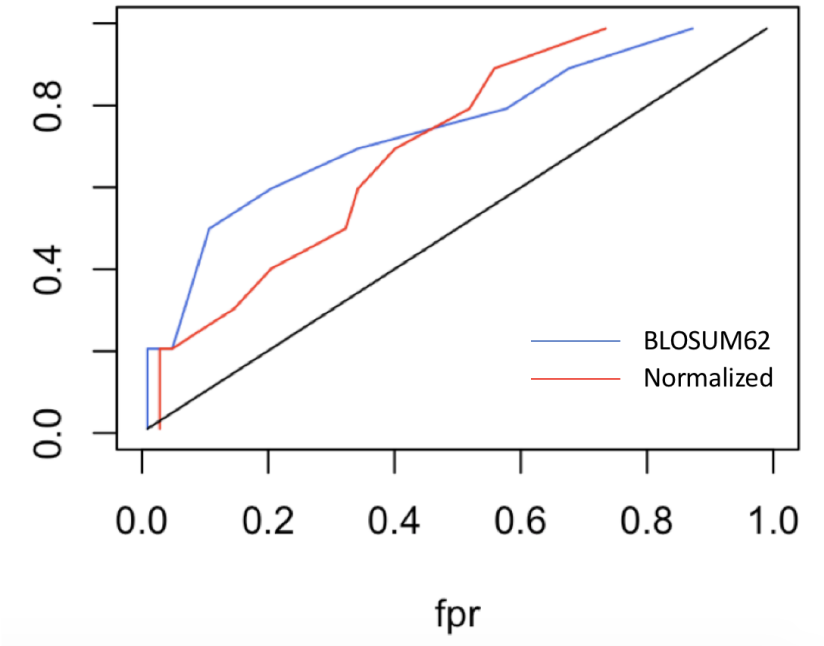


Figure 5: image5

Batch gradient descent, in general, works by predicting a scoring matrix, and assessing how the matrix performs in our error function- the summed_tpr. If the error decreases, the function takes another step in that direction- basically, follows the gradient- until the number of iterations is exceeded or the function hits a minimum/maximum. This algorithm works well in converging on minimums in a function without the need for extra supervision or biological knowledge.

However, a major drawback to this algorithm is that it is very time intensive. It also does not address the constraint that the matrix must be symmetric.

The argument that my optimized scoring matrix would be valuable to the greater biology community would only be valid if I could demonstrate that the matrix was generalizable. That is, I would need to support the claim that the changes I learned to the BLOSUM62 improve the ability to create good alignments for all combinations of amino acids, not just my training set. Given that we have a highly curated set of sequences, I do not think that I would be able to substantiate this claim. A scoring matrix should be general purpose, so that biologists across many experiments can yield comparable results.