Bayesian Statistics: Sequence Matching

Mitchell Kwock

Abstract

In Communications and Biology, matching two sequences with an unknown separation is a constant problem that always needs to be solved. In signals, there is often data that affects itself, dispersing its information in random and varied ways. In Biology, we find that DNA also has sequences that are important to understand the nature of. However, in order to get a good hold of the sequence's meaning, we have to know what the sequence is in the first place.

1. The Project

In order to obtain the readings of a strand of DNA, biologists use PCR machines to copy DNA en masse and use the wonderful process of forward and reverse reading of DNA strands. What these biologists get is something that looks like the following without the line up:

Forward: ATTACGGTAAATCGGATCCCCTGAA.....

Reverse: .....TCGGATCCCCTGAAAAAATCCGAGGATC

However, there are often errors in the reading due to the nature of using small enzymes to pick up on these strands. In reality, we see that as the strand is read, the values are more likely to become incorrect. They become less distinct, more random, and increasingly harder to work with. This is where this Bayesian Genome matcher comes in. By taking the two strands and applying Bayesian updates to each possibility, the program finds the most probable separation distances and attempts to reconstruct the strand.

2. Modeling the Genome

Making a genome to test is actually fairly straightforward: string together a sequence of about 10,000+ base pairs together(a length of 10kb or 10 kilobases). This is the basic string that will be "sequenced" in different parts.

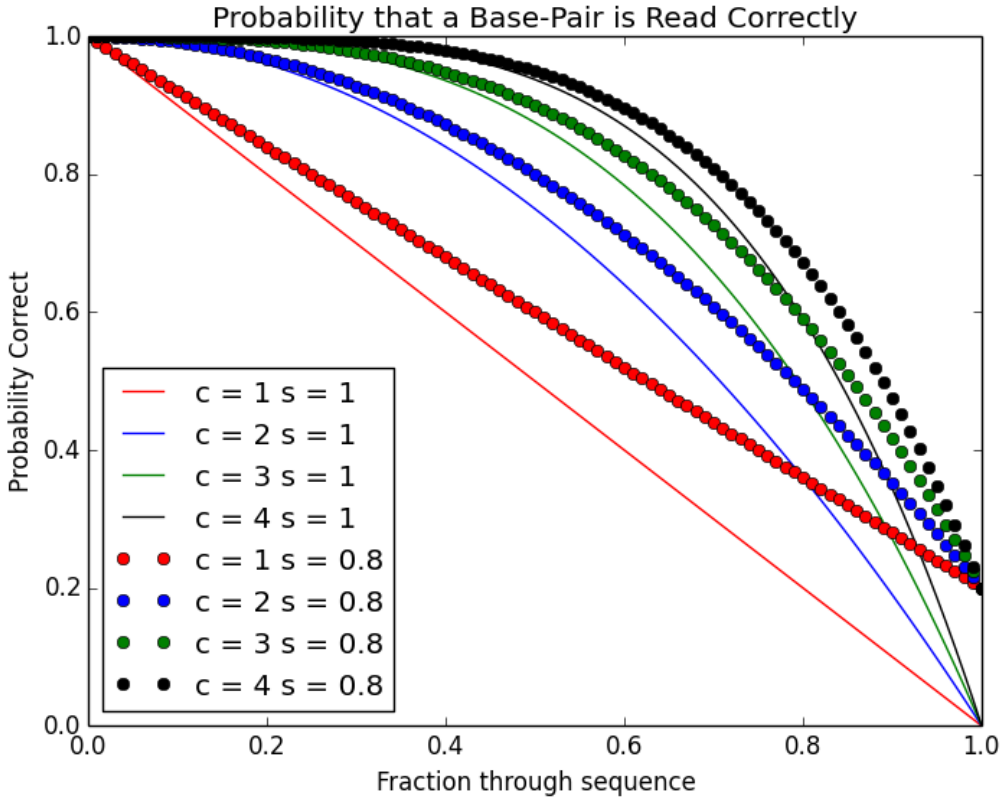
The more interesting problem is modeling the forward and reverse read for some of this sequence. We know a few things regarding the actual nature of reading sequences.

- In the beginning, the sequence has a very high likelihood of being read correctly (That is, an A will likely read as an A as opposed to a G).
- As the sequence progresses, the likelihood of reading correctly decreases and never increases. $\frac{\delta_{Err}}{\delta_{Index}} >= 0$ for all Indices.
- Early to mid reads are consistently good. Only later reads fall prey to randomness.

Using this information, I needed to develop a method to read a genome sequence correctly at times, and incorrectly at others. The model developed by working with the requirements above:

Equation	Explanation
$p = 1, x < 0.5$ or $p = 0, x > 0.5$	Meets the end conditions
$p = 1 - x$	Describes decreasing accuracy
$p = 1 - x^c$	Describes the initial accuracy and the later inaccuracy
$p = 1 - s * x^c$	Gives a baseline accuracy at the end by decreasing the size of s

As seen above, the error handling is done with a simple polynomial function that generally describes what happens when reading the genome and getting a sequence. If it reads incorrectly, then a random A, T, C, or G is selected and put in to that slot.



Interesting aside: When comparing the forward read and the reverse read, the area where the forward read is most reliable is where the reverse read is most unreliable and vice-versa.

3. Modeling the Bayesian Update

It'd be no fun if these were hand-matched! That's where Bayesian comes in. Given each base-pair, we find a lovely and frightening problem. Bayesian updates require us to know one thing to compare our hypotheses: the relative likeliness of the hypothesis based on the data. In this case, we need to find how likely a given sequence shift occurs if a base pair matches or doesn't match.

Hypothesis	The sequences are shifted N base-pairs
Data	At index I, the base pairs either match, don't match, or don't exist

In order to solve this, the problem must be split into the three possibilities and solved for accordingly.

3.1. The Pairs Match

Given that the data matches, let's assume they're both AA. The forward read will have a p chance of being A while the reverse will have a k chance of being A. (This is the same no matter what the match) These two probabilities are calculated using the previously described Probability for reading the base-pair correctly.

Forward: Chances for each base

A	T	C	G
p	$\frac{1-p}{3}$	$\frac{1-p}{3}$	$\frac{1-p}{3}$

Reverse: Chances for each base

A	T	C	G
k	$\frac{1-k}{3}$	$\frac{1-k}{3}$	$\frac{1-k}{3}$

This means that the chances for being positively matched under these circumstances is the dot product of the vectors (That is, the chance that a AA is read plus to the chance that TT is read, and so on):

$$\text{Positive: } p*k+\tfrac{(1-p)(1-k)}{3} \quad \text{False Negative: } 1-[p*k+\tfrac{(1-p)(1-k)}{3}]$$

3.2. The Pairs do not Match

The other case occurs when the original data doesn't match. Perhaps the original bases with these shifts are A and T for forward and reverse respectively. How will this be factored into the final Bayesian update? Time to find out

Forward: Chances for each base

A	T	C	G
p	$\frac{1-p}{3}$	$\frac{1-p}{3}$	$\frac{1-p}{3}$

Reverse: Chances for each base

A	T	C	G
$\frac{1-k}{3}$	k	$\frac{1-k}{3}$	$\frac{1-k}{3}$

Here we get a less pretty, but still workable set of equations:

$$\text{False Positive: } \frac{p+k-2pk}{3} + \frac{2(1-p)(1-k)}{9} \text{ Negative: } 1 - [\frac{p+k-2pk}{3} + \frac{2(1-p)(1-k)}{9}]$$

Given this information, if we read a match, what is the likelihood that the hypothesis is correct? Well, it's the possibility that it was a match and a match was read divided by the chance that there was no match and there was a match read. That is:

$$\frac{Positive}{Positive+FalsePositive}$$

So, if there is no match, what is the likelihood that the hypothesis is correct? The answer is, of course, based on the chance that it is falsely negative divided by the chance that it is read falsely:

$$\frac{FalseNegative}{Negative+FalseNegative}$$

3.3. Index Out of Range Exception!

The last case occurs when an index is out of range, that is, there is nothing to compare the base against. The returned likelihood is always 1 because if there's no index to match against, that is exactly the result one would expect. Therefore this data does not tell us much more information about this hypothesis.

Now we have the beginnings necessary to begin the code's Likelihood function and make it match sequences for us.

4. Execution

This is personally my favorite part of the project: actually getting something to work! Here is the python code described to fit the requirements back in *Modeling the Genome*

```
def Prob_True(portion):
        '''
        Indicates the probability that the returned value is actually a true read.
        portion is the fraction through the string ()

        Requirements:
        Perfect (1) reading at portion=0
        Lowest reading (minimum 0) at portion = 1

        At all points in between: dProb_True/dPortion < 0 (Meaning probability is alu
        '''

        # 1-(p)^c
        scale = s.Get("prob-max") - s.Get("prob-min")
        y_intercept = s.Get("prob-max")
        if(scale > 1 or scale < 0):
                scale = 1
                y_intercept = 1.0

        return y_intercept - portion ** s.Get('curve') * scale
```

Just a quick peek at the Likelihood function that our thinkbayes2 library needs us to define:

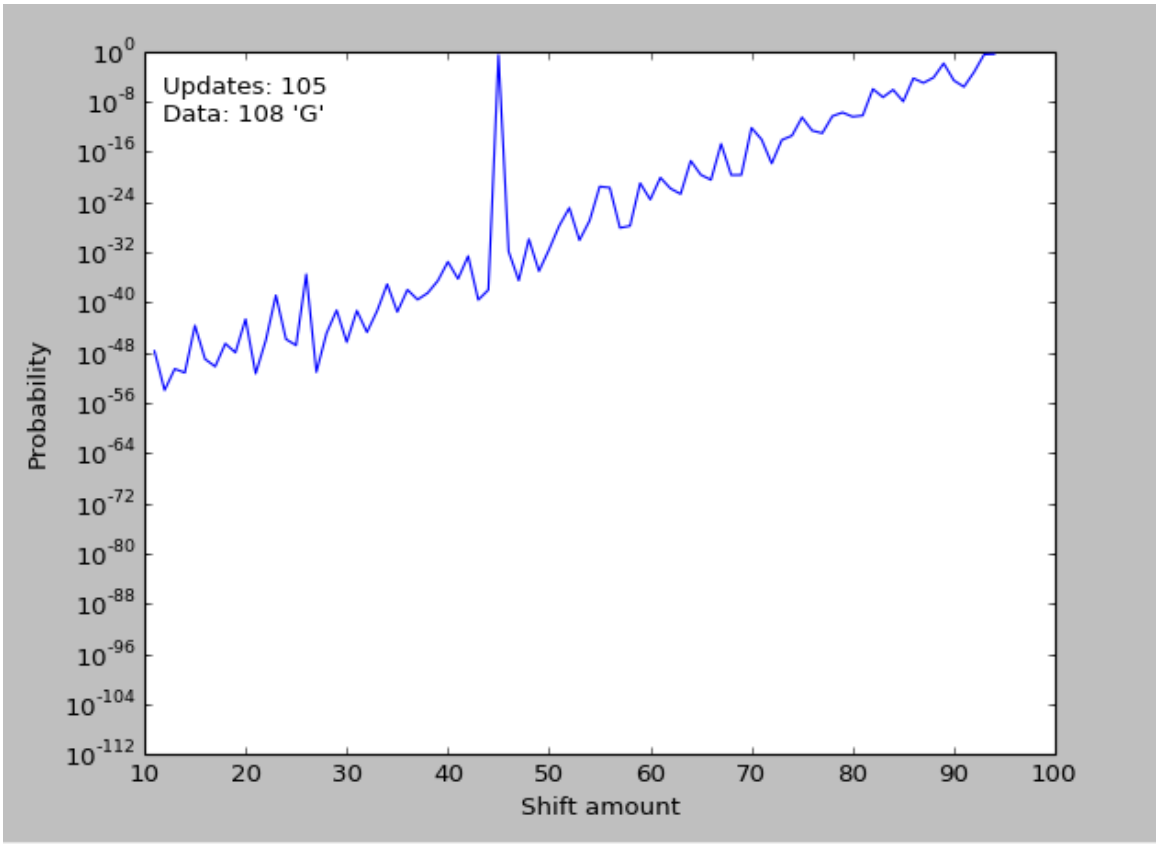
```
def Likelihood(self, data, hypo):
        '''
        hypo: Starting index of the intended string
        data: [Index, reverse_base] The read Base at Index

        Returned probability is the probability that they were matched
                Needs to account for:
        Increasing error in forward string as well as decreasing in reverse
        '''
        index        = data[0]
        reverse_base = data[1]
        index_match  = hypo + index

        # Deals with the no-match case because index is out of range
        if(index_match == self.forward_length):
```

```
        # Most probably true at later values
        return(1.0 * index / self.min_length)
    if(index_match > self.forward_length):
        return 1

    forward_base = self.forward_string[index_match]

    if(forward_base == reverse_base):
        # Match Found, determine chance of match accounting for False Positiv

        # Accounts for random chance of reading correctly because of weighted
        # Forward Probability of reading correctly

        f_portion = 1.0 * index_match / self.forward_length
        p = (1 + 3 * Prob_True(f_portion))/4
        # Reverse Probability of reading correctly
        r_portion = 1 - 1.0 * index / self.reverse_length
        k = (1 + 3 * Prob_True(r_portion))/4

        positive = p * k + (1-p)*(1-k)/3
        false_positive = (p + k - 2*p*k)/3 + (1-p)*(1-k) * 2/9
        return positive / (positive + false_positive)
    else:
        # No match found, what is the possibility that it's a false negative?
        # Forward Probability of reading correctly
        f_portion = 1.0 * index_match / self.forward_length
        p = (1 + 3 * Prob_True(f_portion))/4
        # Reverse Probability of reading correctly
        r_portion = 1 - 1.0 * index / self.reverse_length
        k = (1 + 3 * Prob_True(r_portion))/4

        false_negative = 1 - (p * k + (1-p)*(1-k)/3)
        negative =          1 - ((p + k - 2*p*k)/3 + (1-p)*(1-k) * 2/9)
        return false_negative / (negative + false_negative)
```

Notice that there is the case handling for the cases. The possibility that the index is too long is actually based on the chance of said data not being seen. The chance that the 100^{th} index in a sequence of 101 is not seen is actually very high, however, the chance of the 10^{th} index in a sequence in 101 is fairly low, lowering that hypothesis's chance of being correct.

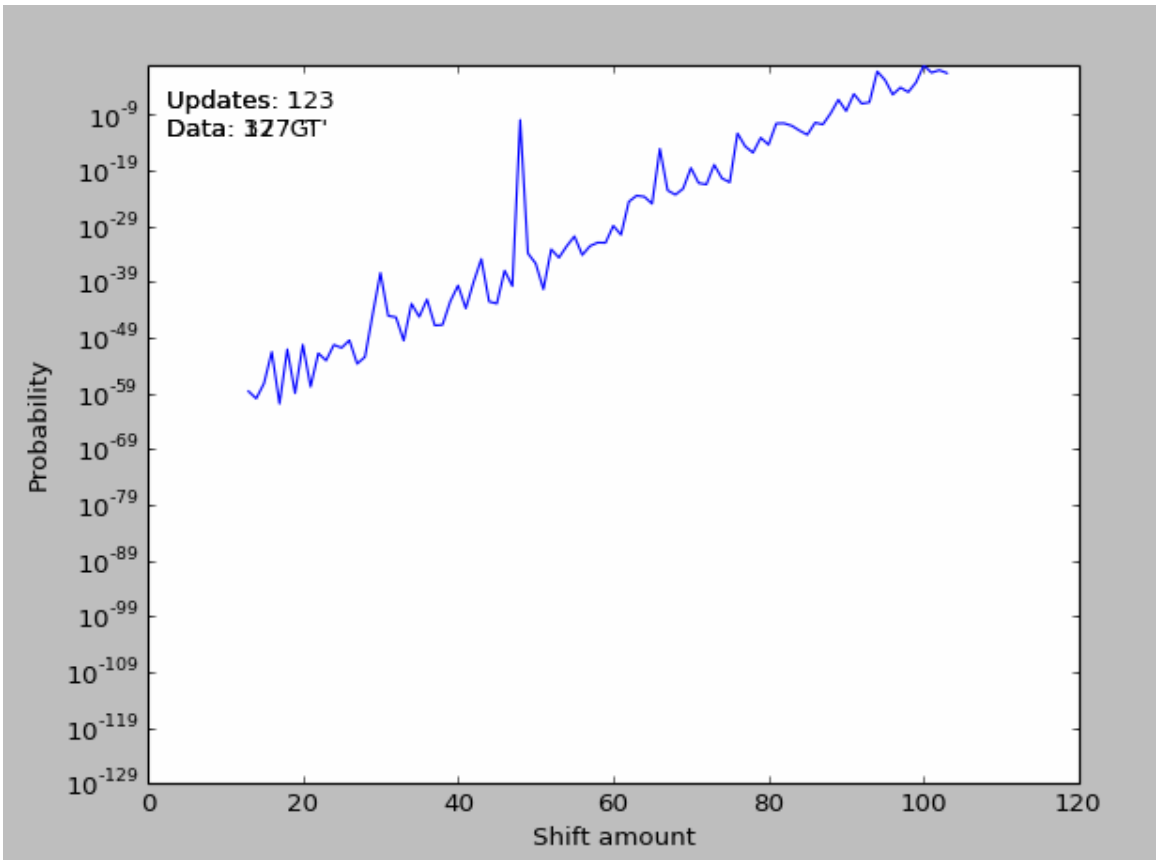
5. Results

The results are actually fairly useful and brings to light things that I did not expect!



By iterating through the data, we learn some other fun things about what Bayesian probability tells us: Lack of information can sometimes point us in a better direction. We see a single spike, the actual data, but the general idea is that with more wrong data, the end piece is more and more likely compared to everything else (Fewer mistakes by virtue of being at the end). If, for instance, there were sufficient incorrect matches, a later value will be more likely because there are fewer WRONG values

Let's look at another execution, this time executing falsely:



The correct shift is 44, as we can see in the above, but because there were a sufficient number of incorrect bases, it is actually much lower (it's *about* 10^{-10}) than the outputted "best" outcome of 100. Graphing this on a Log Scale gives us much more insight as to the outcome of the probability. Even though I know the outcome is wrong, the probabilities given to the engine are all correct. This tells me that based on the sheer unlikelihood of that many bases being correct outweighs the matches sufficiently. This is unacceptable!

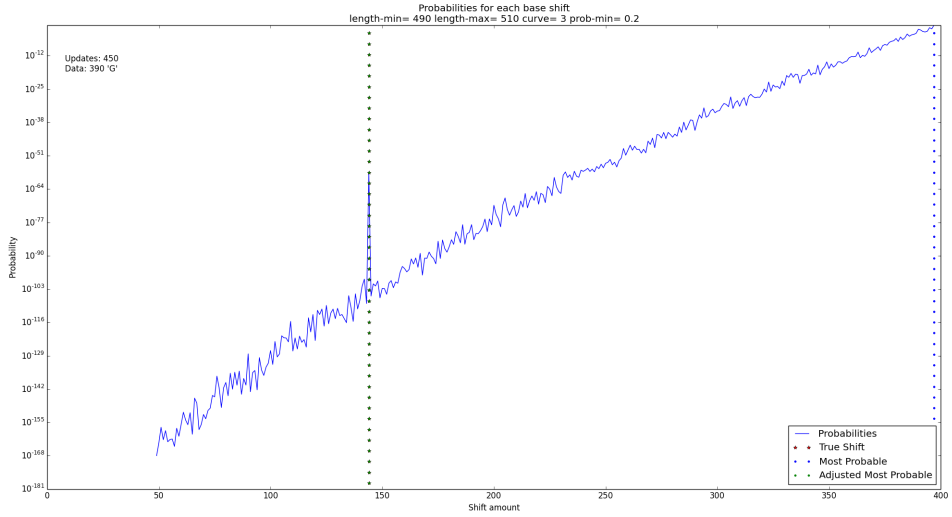
6. Recognizing These Outliers

In order to actually make a reliable sequence matcher, I cannot have the correct choice blotted out simply because it doesn't match perfectly (And even if it did, being later simply does not decrease the chances enough) That is why another iteration of the code is in order.

To the human eye, the outlier is obvious: The spike in the function. Recognizing that the rest of the function is roughly linear, taking a linear regression of the logarithmic function gives us a line, and the adjusted pick is the one whose value is highest above the line. In order to model this "Human Pick", the following code has been added to test:

```
ex = np.asarray(list(suite.Values()))
why = np.log([suite[x] for x in suite.Values()])
slope, intercept, r_value, p_value, std_err = stats.linregress(ex, why)
adjusted = why - slope * ex
adjusted_high = list(adjusted).index(max(adjusted)) + suite.Values()[0]
```

This extra code finds the location where the shift is highest relative to what it should be, that is, the spikes in the graph. Running the code again yields the following results:



That's much better! By adding this smart-picker, as long as there is a spike, that shift is chosen. This spike is also much more likely than its neighbors for a reason. This method is particularly useful when comparing large sequences because even if every base matched perfectly, the stacking slight improbabilities of the data mean that the final probability for a shift will be ever decreasing.

7. Looking Forward

This was a fun exercise in Bayesian Statistics and a reminder to the the mind-blowing nature of Bayesian updates. The fact simply is, given the probabilities of things, coming later is more likely than the incorrectness of the various bases. I feel the need to optimize the code to pick out the *correct* shift more often, but the simple fact is, if I want to make the proper choice, I must believe in the math.

To extend this project, there are several things that can be done:

- Add weighting to the different bases, changing their individual probabilities.
- Experiment and change the Probability function to included added/shifted bases.
- Extend the overlap such that the entire forward and reverse read are contained within one another.
- Have a complete probability set for the reads.
- Add visualization tools.