MBI Homework 2 for CS students

Autor: Marián Kravec

a)

Firstly we implement functions that get two bases and time, and returns probability that first base change to second in that time based on Jukes-Cantor model:

```
def mutation_prob(base_a, base_b, alfa, t):
    if base_a == base_b:
        return (1+3*np.exp((-4*alfa*t)/3))/4
    else:
        return (1-np.exp((-4*alfa*t)/3))/4
```

Now we can implement Falsenstein's algorithm (I am not sure whether my implementation can be considered as dynamical programming or it's recursive with memoization but results are equal):

```
def falsenstein_one_col(root, tree, alfa, leaf_bases, poz):
    sol_table = dict()
    total_p = 0
    nodes = tree.keys()
    def tree_prob(root, base):
        sol = sol_table.get((root, base), None)
        if sol is not None:
            return sol
        if root not in nodes:
            leaf_val = leaf_bases[root][poz]
            if leaf_val == 'N' or leaf_val == '-':
                return 1
            elif base == leaf_val:
                return 1
            else:
                return 0
        else:
            childrens = tree[root]
            total = 1
            for child, time in childrens.items():
                temp = 0
                for child_base in bases:
                    temp += (tree_prob(child, child_base)
                             *mutation_prob(base, child_base, alfa, time))
                total *= temp
            sol_table[(root, base)] = total
            return total

    for base in bases:
        total_p += tree_prob(root, base)
    return total_p/4
```

b)

If we run this algorithm on tree stored in tree.txt, setting all leaf bases to A and setting mutation speed to $\alpha = 1$ and $\alpha = 0.2$ we get these values:

α	probability (float)	probability (percentage)
1	0.054	5.4%
0.2	0.183	18.3%

(I got bit confused by information in assignment "The probability of each base in the equilibrium is $\frac{1}{4}$, which is used as probability q_a of generating base a in the root"whether this information means that information means that I need to divide probability in root by four or not, after discussion with some classmates I did exactly that (divide by four) and got values in table above instead of values 21.8% for $\alpha = 1$ and 73.3% for $\alpha = 0.2$)

We can see that smaller mutation speed resulted in higher probability. This makes sense because fact that all bases are same means either that no mutations happened or that mutations happened but also 'inverse' mutations happened that returned bases back to initial state. Out of these two cases no mutations is more likely scenario however likelihood of this scenario decrease as mutation speed increase but probability to not change decrease.

c)

Now we implement function that can give us probability of whose sequence rather that one column. This function compute probability for each column and because we expect that columns are independent we multiply those values (to not have issues with very small numbers we sum logarithms of probabilities for columns and in the end we compute it back to probability value by putting it as exponent).

```
def falsenstein_all_col(root, tree, alfa, leaf_bases, start, length):
    total_log_p = 0
    #k = len(list(leaf_bases.values())[0])
    for i in range(start, start+length, 1):
        col_prob = falsenstein_one_col(root, tree, alfa, leaf_bases, i)
        if col_prob == 0:
            total_log_p -= float("inf")
        else:
            total_log_p += np.log(col_prob)
    return np.exp(total_log_p)
```

We also implement function that run Falsenstein's algorithm on inputted tree for 20 values of mutation speed α (between 0.1 to 2 with step 0.1) and return one for which Falsenstein's algorithm returned highest probability.

```python
def opt_mut_speed(root, tree, leaf_bases, start, length):
    test_alfa = 0.1
    best_alfa = 0.1
    best_log_prob = -float("inf")
    for i in range(20):
        test_prob = falsenstein_all_col(root, tree, test_alfa, leaf_bases, start, length)
        if test_prob == 0:
            test_log_prob = -float("inf")
        else:
            test_log_prob = np.log(test_prob)

        if test_log_prob > best_log_prob:
            best_alfa = test_alfa
            best_log_prob = test_log_prob
        test_alfa =round(test_alfa+0.1,1)
    return best_alfa
```

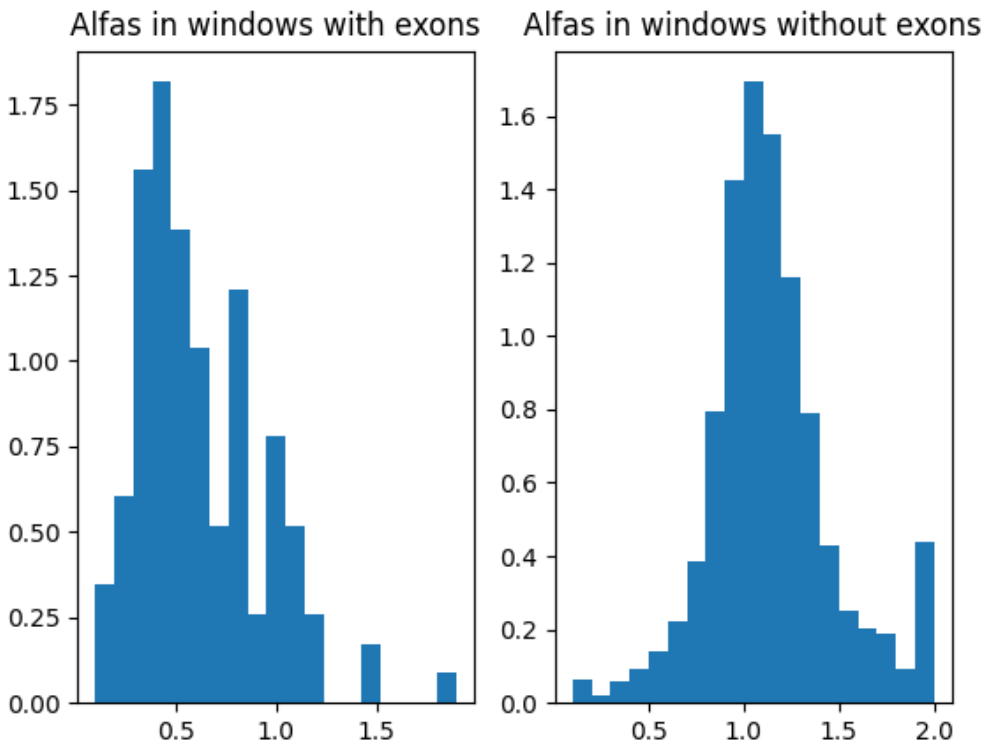
d)

Now we will read file cftr.txt which contains aligned sequences for animals in leafs of our tree. Now we will split this sequences into non-overlapping windows with size 100 and for each window we compute most likely value of mutation speed α using function we implemented in previous section. For first 10 windows we get these values $\alpha*$:

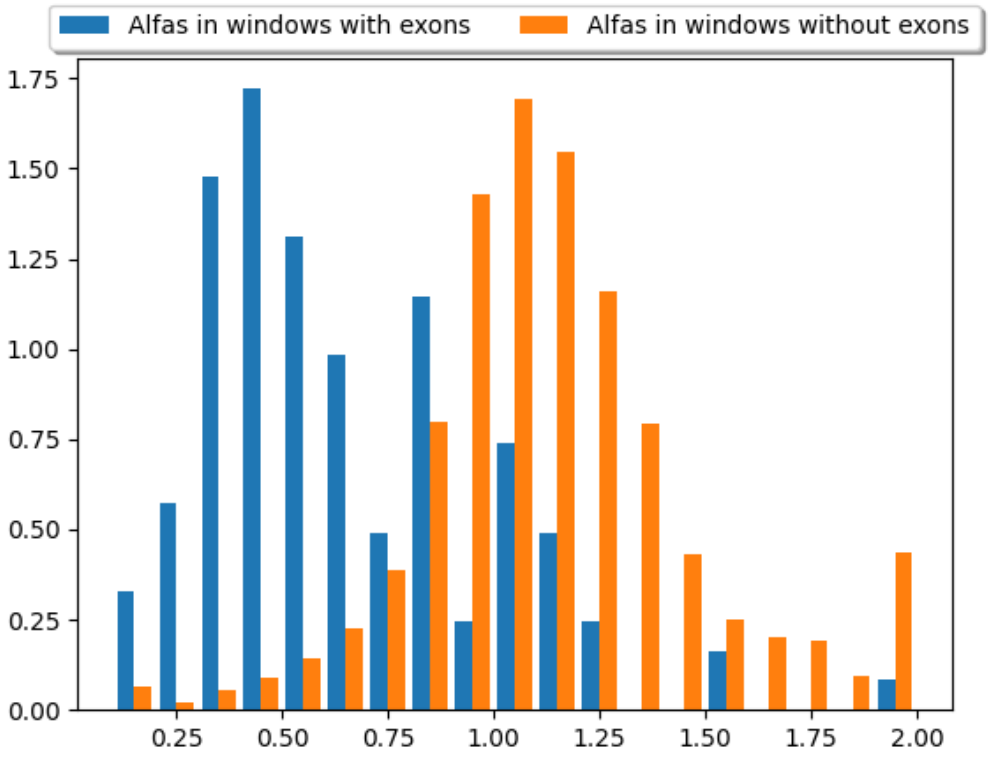
position	$\alpha*$
1-100	1.0
201-200	1.2
301-300	1.1
401-400	2.0
501-500	1.6
601-600	1.0
701-700	1.5
801-800	1.3
901-1000	2.0
1001-1100	1.1

e)

After we compute $\alpha*$ for all windows we split them into groups that contain at least one base from exons and into those that are contain only bases from introns (for that we will use file exons.txt). After split we can visualize their distribution using histogram (we used density view of histogram for better comparison because windows that have overlap with exon are only small fraction of all windows):



We can see that their distribution is quite different. Let's put them both into one histogram for better comparison:



In this histogram we can clearly see difference. We can see that optimal value α for windows that contains at least one base from exon is generally smaller than optimal value for windows containing only intron. This is expected because lower mutation speed means that there are less mutations in given time and mutation in exons are much more dangerous then in introns. It's more dangerous because exons are parts of genome that gets translated into protein and change in one base means change in codon which could mean change in amino acid which could change structure and functionality of resulting protein. On the other hand changes in introns have smaller consequences and because of that their mutation speed can be higher so mutations in them are more frequent.