CS415 - Project 3 - Ethan Corgatelli

Title:	Parameter Set Estimation (Generating Scoring Matrix, Emission Probabilities and Transition Probabilities from an Aligned and Labeled Sequence Set)
Date:	04/28/2023

1.0 Abstract

This paper describes how I completed Project 3. With Google Collab and Python I used the provided data files to calculate an alignment scoring matrix, emission probabilities for each label, and transition probabilities between the labels. I did this without defining any functions or structures of my own, instead using only variable assignments and nested *for* loops for value calculation. Ultimately this is not great coding practice, but I decided that for this project it would be best to keep everything simple and readable rather than adding extra boilerplate code. I believe that I have a firm understanding of both the theory and my method of implementation. I've also noticed that the results of my code match my intuition of what they should be. I have also selected a *LAMBDA* value (0.25) that generates alignment scores that are similar to those of the Blosum50 scoring matrix. For these reasons, I trust my parameters and methods, and therefore I also trust my results.

My code can be found here: http://tiny.cc/cs415-corg-p3-code

2.0 Algorithm Description

This section contains a complete description of how my program generates the required parameter estimations. First it reads in the first data file using *open()*. Then it uses list comprehension to create the variable *pop* which stores the lines of the data file in a list (with whitespaces removed). Another variable, *pop_raw*, is made by concatenating the elements of *pop* into a single string, which is helpful for calculating background frequencies. A third variable (*pop_columns*) stores the population as a list of the columns instead of a list of the rows (like *pop*), which is helpful for counting pairs of amino acids.

Once the data had been read in, I used *set()* then *list()* then *sort()* on *pop_raw* to get a list of all the amino acids in the data set. I then copied-pasted the list and stored it as the constant *ALPHABET*, then deleted the code I used to generate it.

The final results (as well as most of the intermittent calculations) of my program are stored as dictionaries (or a list of dictionaries), as they are quite fast as well as easy to use.

2.1 Scoring Matrix

Several variables must first be calculated in order to calculate the final scoring matrix (which is a dictionary called *align_score*): *bg_freq* (a dictionary), *pair_counts* (a dictionary), *POSSIBLE_MATCHES* (a constant integer), and *target_freq* (a dictionary). All three of the dictionaries have the exact same set of keys, which are generated with a doubly-nested *for* loop, where the outer loop iterates over all the characters in *ALPHABET*, and the inner loop iterates over the characters in ALPHABET starting with the character of the outer loop going through to the end of the list. Inside the inner loop, the characters from each of the loops are concatenated and added as a key to the dictionary (with a value of 0). This effectively leaves us with an empty triangle matrix for every possible pairing of amino acids (where the alphabetically first letter in the key always comes first), without duplication (i.e.: *'aa', 'ac', 'ad',* … , *'ay', 'cc', 'cd',* … , *'cy', 'dd',* …). See **Section 3.1** to better understand the dictionary structure being used.

bg_freq stores the background frequencies at which each amino acid appears in the data set. It is calculated using a *Counter* (from the *collections* library) on *pop_raw* to get the frequencies of each amino acid, then converting it to a dictionary and dividing each value by the length of *pop_raw* to get the final background frequencies.

pair_counts stores the observed counts of pairs of amino acids. It is generated using a triply-nested *for* loop. The outside loop iterates over *pop_columns*, whereas the inside loops mimic the behavior of the doubly-nested *for* loop that was used to create the dictionaries (iterating over the amino acids in <u>each of the columns of the population</u>, instead of iterating over the <u>list of possible amino acids</u>). Inside the inner loop, the currently observed pair is used to generate the correct key (of *pair_counts*) whose value should be incremented (and it gets incremented).

POSSIBLE_MATCHES is the maximum number of times than any two amino acids could possibly appear paired (or in other words, how many pairs of amino acids are there in the data set in general?), which is calculated with the formula *(pop_len * (pop_len - 1) / 2) * STRAND_LEN* (where *pop_len* is the number of individuals in the data set).

target_freq stores the relative frequencies of each possible pair of amino acids, calculated by dividing each value in *pair_counts* by *POSSIBLE_MATCHES*.

Finally, ***align_score*** stores the final alignment scores, calculated with the formula *round((math.log(target_freq[a + b] / (bg_freq[a] * bg_freq[b]))) / LAMBDA)*.

2.2 Emission Probabilities

In order to calculate emission and transition probabilities, the second data file is read and every third line (which is the label data) is added to a list called *labels*. Similarly to the original data set, a second variable, *labels_raw* (a single string containing all the labeling data), is stored. Two lists (*e_freq* and *l_freq*) are used in the calculation of the final emission probabilities, stored in *e_prob*, which is a list of three dictionaries (one for each label).

e_freq is also a list of three dictionaries, counting the frequency at which each amino acid is emitted from each of the three labels.

l_freq is a list of three integers, counting the frequency at which each label appears in the data set.

Finally, ***e_prob*** is calculated by dividing all values of each dictionary in *e_freq* by the appropriate label's frequency (from *l_freq*), which yields the final list of dictionaries containing the probabilities for emitting each amino acid at each label.

2.3 Transition Probabilities

The transition probabilities, like the scoring matrix and emission probabilities, are stored in a dictionary (called *t_prob*). The keys are strings of length two, where the first character is the 'from' state and the second character is the 'to' state. The calculation of transition probabilities requires the calculation of two intermittent variables: *t_freq* and *t_bg_freq*.

t_freq is a dictionary that counts the amount of times each possible transition takes place, and is calculated by iterating over each string in *labels* (see **Section 2.2**). For each string, the program first creates a variable called *pl* (for 'previous label') and sets it to *None*, then iterates over the characters in the string. If *pl* is still *None*, then it is set to the current (first) character. On subsequent executions of the loop, the correct key (of *t_freq*) is generated using *pl* (the previous label) and the current label. That key's value is incremented.

t_bg_freq is a list of 3 integers that counts the number of transitions out of each of the tree states by adding up the appropriate entries from *t_freq*.

Finally, ***t_prob*** is calculated by dividing each entry in *t_freq* by the appropriate value from *t_bg_freq* (which is the state from which that entry transitioned).

3.0 Results

This section contains the parameter sets that my program generated from the data sets that were provided.

3.1 Scoring Matrix

The first and last pairing from each first-letter-separated section is highlighted in order to more easily observe the triangular nature of the scoring dictionary.

```
{'aa': 10,'ac': 2, 'ad': 5, 'ae': -1,'af': -3,'ag': -3,'ah': 2,
 'ai': -3,'ak': -1,'al': -3,'am': -1,'an': 2, 'ap': -2,'aq': -1,
 'ar': -2,'as': -4,'at': 5, 'av': 2, 'aw': 2, 'ay': -2,'cc': 6,
 'cd': 4, 'ce': 1, 'cf': 2, 'cg': 2, 'ch': 2, 'ci': 1, 'ck': 4,
 'cl': 4, 'cm': 2, 'cn': 1, 'cp': 2, 'cq': 0, 'cr': 1, 'cs': 3,
 'ct': 1, 'cv': 2, 'cw': 0, 'cy': 4, 'dd': 5, 'de': 4, 'df': 3,
 'dg': 2, 'dh': 2, 'di': 2, 'dk': 3, 'dl': 1, 'dm': 1, 'dn': 1,
 'dp': 2, 'dq': 1, 'dr': 1, 'ds': 3, 'dt': 3, 'dv': 2, 'dw': 2,
 'dy': 2, 'ee': 6, 'ef': 2, 'eg': 4, 'eh': 4, 'ei': 4, 'ek': 1,
 'el': 0, 'em': 3, 'en': 2, 'ep': 1, 'eq': 2, 'er': 3, 'es': 3,
 'et': 1, 'ev': 0, 'ew': 3, 'ey': 2, 'ff': 5, 'fg': 3, 'fh': 2,
 'fi': 4, 'fk': 0, 'fl': 2, 'fm': 3, 'fn': 1, 'fp': 3, 'fq': 3,
 'fr': 4, 'fs': 2, 'ft': 2, 'fv': 2, 'fw': 2, 'fy': 1, 'gg': 6,
 'gh': 1, 'gi': 3, 'gk': 2, 'gl': 3, 'gm': 2, 'gn': 1, 'gp': 3,
 'gq': 1, 'gr': 2, 'gs': 4, 'gt': 1, 'gv': 1, 'gw': 3, 'gy': 0,
 'hh': 6, 'hi': 3, 'hk': 2, 'hl': 2, 'hm': 1, 'hn': 2, 'hp': 0,
 'hq': 3, 'hr': 2, 'hs': 3, 'ht': 4, 'hv': 0, 'hw': 1, 'hy': 3,
 'ii': 5, 'ik': 2, 'il': 1, 'im': 4, 'in': 3, 'ip': 3, 'iq': 3,
 'ir': 2, 'is': 3, 'it': 2, 'iv': 0, 'iw': 2, 'iy': 1, 'kk': 5,
 'kl': 1, 'km': 2, 'kn': 2, 'kp': 3, 'kq': 2, 'kr': 3, 'ks': 1,
 'kt': 4, 'kv': 2, 'kw': 2, 'ky': 4, 'll': 7, 'lm': 2, 'ln': 1,
 'lp': 3, 'lq': 4, 'lr': 1, 'ls': 3, 'lt': 1, 'lv': 1, 'lw': 2,
 'ly': 3, 'mm': 5, 'mn': 2, 'mp': 1, 'mq': 2, 'mr': 4, 'ms': 2,
 'mt': 1, 'mv': 1, 'mw': 3, 'my': 3, 'nn': 6, 'np': 4, 'nq': 2,
 'nr': 3, 'ns': 1, 'nt': 2, 'nv': 2, 'nw': 2, 'ny': 1, 'pp': 6,
 'pq': 1, 'pr': 3, 'ps': 3, 'pt': 0, 'pv': 1, 'pw': 2, 'py': 1,
 'qq': 6, 'qr': 1, 'qs': 2, 'qt': 3, 'qv': 0, 'qw': 1, 'qy': 2,
 'rr': 6, 'rs': 2, 'rt': 2, 'rv': 2, 'rw': 2, 'ry': 2, 'ss': 5,
 'st': 2, 'sv': 2, 'sw': 2, 'sy': 1, 'tt': 5, 'tv': 1, 'tw': 2,
 'ty': 3, 'vv': 6, 'vw': 2, 'vy': 1, 'ww': 5, 'wy': 1, 'yy': 6}
```

3.2 Emission Probabilities

The dictionaries in this list are in order of the label they belong to (0 then 1 then 2).

```
[{'a': 0.03909590714722053,   'c': 0.03836285888821014,
  'd': 0.018692730604764812,  'e': 0.03909590714722053,
  'f': 0.06597434331093463,   'g': 0.03726328649969456,
  'h': 0.0365302382406841 8,   'i': 0.029077580940745265,
  'k': 0.032864996945632254,  'l': 0.03481979230299328,
  'm': 0.046670739156994503,  'n': 0.055833842394624314,
  'p': 0.05839951130116066,   'q': 0.05436774587660354,
  'r': 0.028222357971899818,  's': 0.05766646304215028,
  't': 0.0596212583995113,    'v': 0.11350030543677458,
  'w': 0.074282223579719,     'y': 0.07965791081246182},

  {'a': 0.0793564221434415,   'c': 0.02754295064085083,
  'd': 0.017180256340332697,  'e': 0.02617943823288792,
  'f': 0.04963185164985001,   'g': 0.018543768748295608,
  'h': 0.01772566130351786,   'i': 0.021270793564221433,
  'k': 0.023997818380147258,  'l': 0.05263157894736842,
  'm': 0.04281428961003545,   'n': 0.023452413416962095,
  'p': 0.037087537496591216,  'q': 0.0218161985274066,
  'r': 0.01745295882192528,   's': 0.04172347968366512,
  't': 0.1109899100081810 8,   'v': 0.14535042268884646,
  'w': 0.11862557949277339,   'y': 0.10662667030269976},

  {'a': 0.015887388870595983, 'c': 0.06857095818241686,
  'd': 0.04181758314125782,   'e': 0.05622324662495884,
  'f': 0.05548238393151136,   'g': 0.04856766545933487,
  'h': 0.03630227197892657,   'i': 0.050131708923279554,
  'k': 0.04955548238393151,   'l': 0.042229173526506424,
  'm': 0.04790912084293711,   'n': 0.0540829766216661 2,
  'p': 0.057540335857754366,  'q': 0.05375370431346724,
  'r': 0.051778070464273955,  's': 0.04897925584458347,
  't': 0.05745801778070464,   'v': 0.053836022390516956,
  'w': 0.06429041817583141,   'y': 0.045604214685544944}]
```

3.3 Transition Probabilities

The first and second letters in the keys represent the ‘from’ and ‘to’ state respectively.

```
{'00': 0.9610645638245441, '01': 0.03893543617545589, '02': 0.0,
 '10': 0.0, '11': 0.9200988467874794, '12': 0.07990115321252059,
 '20': 0.015362896528815811, '21': 0.0, '22': 0.984637103471184}
```