Use the K-mer Method and k^{th}-order Markov Model to classify a set of short genomic sequences

YuqiaoGong Runhan li Ziyun Chen Haoyuan Qiu

Stochastic Process Projrct Report

Date: May 15, 2022

Abstract

The non-aligned method can accelerate the classification of metagenomics[1–3]. In this study, we have implemented two non-aligned methods to classify short reading data. In our research, k^{th} order Markov model is much better than k-mers model in terms of time cost and accuracy of training data. Therefore, we use kth sequential Markov model to allocate test files. The optimization method of k-mers and the model with advanced algorithm are discussed. In addition, the performance of GPU is further considered. All models and codes are available in our github repository.

Keywords: Metagenomics classifiction, Kth Markov model, KMM, K-mers, GPU

1 Problem Restatement

The goal of this project is to use the kth-order Markov Model to classify a set of short genomic sequences. Each short sequence will be classified to one group, which is represented by a genome in a reference genome data set. The file containing short genomic sequences can be found at /home/faculty/ccwei/courses/2022/pab/proj1/reads.fa. The 10 genomes can be found under the directory /home/faculty/ccwei/courses/2022/pab/proj1/genomes/. A test set has been created for you: 20,000 short reads are given in /home/-faculty/ccwei/courses/2022/pab/proj1/test/test.fa. The mapping of aligned genomes and the short sequences listed in test.fa is given in a file called seq_id.map file. You can use these files to test your own script.

2 Method

2.1 K-mers method

K-mers are short words of length K in a gene sequence that can characterize metagenomics. Therefore, in metagenomic classification, K-mers are always taken instead of aligned[4], which would be unacceptably

slow. Here, we use a simple statistical method to classify a given short-read sequence. In our k-mers algorithm, we initialize an N × 4^k matrix *G*, where N represents the number of genomes. Then we convert the DNA sequence of A, T, C, G to the number sequence of 0, 1, 2, 3 using an encoder correspondingly. Note that non-A, T, C or G characters (eg ambiguous IUPAC characters) are ignored in this encoder. For each k-mer, we can get a unique k-bit quaternion, which is then converted to a decimal number as an index into *G*. After these preprocessing, all genomes are input and the probabilities of all k-mers are calculated and stored in *G*. Do the same for allocated short reads. Here, we can get a 4^k vector *l* representing the probability distribution of k-mers in short reads. Methods to measure the distance from *l* to G_i are various, here we use Manhattan Distance(Eq1), Euclidean Distance(Eq2), Pearson Correlation Coefficient(Eq3) and Cosine(Eq4) as the standard, which can estimate the k-mer similarity between a specific genome and short reads. We choose the shortest d_i as the result of the assignment. we will also discuss about the efficiency of these measurement later.

$$d_i^M = |l - G_i| = \sum_{j=0}^{4^k-1} |l_j - G_{ij}| \tag{1}$$

$$d_i^E = ||l - G_i|| = \sqrt{\sum_{j=0}^{4^k-1} (l_j - G_{ij})^2} \tag{2}$$

$$d_i^P = \frac{Cov(l, G_i)}{\sigma_l \sigma_{G_i}} \tag{3}$$

$$d_i^C = \frac{lG_i}{|l||G_i|} = \frac{\sum_{j=0}^{4^k-1} l_j \times G_{ij}}{\sqrt{\sum_{j=0}^{4^k-1} l_j^2} \sqrt{\sum_{j=0}^{4^k-1} G_{ij}^2}} \tag{4}$$

2.2 K^{th}-order Markov Method

In the model used by the k-order Markov method, the probability of an event occurring depends not only on the current state, but also on the k recent past state transitions[5]. The advantage of this approach is that automation is possible, and one can utilize more prior information in the past to get a better prediction in the future, which makes sense in genome classification. In a first-order Markov model, each state depends only on past states, which can be described by a 4 × 4 transition probability matrix M, where 4 represents the basis A, T, C, G, and a_{ij} represents the transition probability from the matrix in the left base i to the right base j. For example, a_{14} represents the transition probability from A to G.(Eq 5)

$$M_1 = \begin{array}{c} \\ A \\ T \\ C \\ G \end{array} \begin{array}{c} \begin{array}{cccc} A & T & C & G \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \end{array} \tag{5}$$

In a k-order Markov model, when k is determined in advance, each state depends on the past k states. The transition probability can be obtained by equation 6:

$$kMM_{i,mn} = P(O_m|O_n) = \frac{F_i(O_m|O_n)}{F_i(O)} \tag{6}$$

where O_m and O_n are oligonucleotides of length k, $P(O_m|O_n)$ represents the probability of transition from O_m to O_n, and $F(O_m|O_n)$ represents the observed count of transitions from O_m to O_n in genome sequence i, $F(O_m)$ is the observed count of O_m. Therefore, we can build a new matrix for the k-order Markov model:

$$M_k = \begin{array}{c} \\ O_1 \\ \vdots \\ \vdots \\ O_{4^k} \end{array} \begin{array}{c} \begin{array}{cccc} O_1 & \cdots & \cdots & O_{4^k} \end{array} \\ \begin{pmatrix} b_{11} & \cdots & \cdots & b_{14^k} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ b_{11} & \cdots & \cdots & b_{14^k} \end{pmatrix} \end{array} \tag{7}$$

There are 4^k rows and 4^k columns in the matrix, representing all possible permutations of the past k states. The transformation from O_m to O_n can actually be reduced to from O_m to A, T, C, G, since the only change occurs on the last base of O_n. Then we can rewrite our model as a $4^k \times 4$ transition matrix.(Eq 8) As for our problem, there are 10 genomes in total, so we will have 10 transition matrices. The query sequence score for reference genome i is given by Eq 9

$$M_k = \begin{array}{c} \\ O_1 \\ \vdots \\ \vdots \\ O_{4^k} \end{array} \begin{array}{c} \begin{array}{cccc} A & T & C & G \end{array} \\ \begin{pmatrix} a_{11} & \cdots & \cdots & a_{14} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{11} & \cdots & \cdots & a_{14} \end{pmatrix} \end{array} \tag{8}$$

$$S_i = -\sum_{j=0}^{l-k-1} ln(P_i(O_j|O_{j+1})) \tag{9}$$

where O_j and O_{j+1} are two oligonucleotides of length k, and $P(O_j|O_{j+1})$ is the observed transition probability from O_j to O_{j+1} in the i_{th} genome. l is the length of the query sequence. Each short read is then compared to 10 matrices, and we can take the matrix with the lowest score as the reference genome.

2.3 GPU acceleration

2.3.1 Basic principles of GPU

In this experiment, we use Python numba for CUDA Programming to realize the manipulation of GPU parallel computing. Numba is an open source JIT compiler for Python, developed by Anaconda company, which can accelerate the CPU and GPU of Python native code. In addition, numba is very friendly to numpy arrays and functions.

The basis of GPU processing is that each thread should be independent of other threads. The problem that arises here is that the K-mer vector or KMM matrix describes the overall distribution of the sequence, and the computation results of each thread need to be aggregated to get the final vector or matrix. The solution is, in the first GPU function, compute the index of each k-mer and store it in a list. Next, count index frequency Without going through the entire index list, we divide this large loop into many small loops, each thread in the GPU processes a sub-loop, and then aggregates their results in the CPU.

When we make predictions, each read is independent of the others, which makes it absolutely suitable for sorting each read into its owning genome in parallel with GPU capabilities. For the K-mer method, the classification requires the read's K-mer vector, but due to the limited Python support of Numba's CUDA, it is inconvenient to compute the K-mer vector in parallel in the GPU kernel function. Therefore, we encode and compute the read K-mer vectors ahead of time without GPU. For the KMM method, the entire classification process can be parallelized because for each encoded sequence, we can directly calculate its probability from the KMM matrix. In each thread, we compare the read data with 10 genomes and get the best classification result. Reverse complement sequences are also taken into account.

2.3.2 grid step technology

With the help of the GPU, we can process multiple positions at once, instead of going through the entire sequence, counting every k consecutive bases in turn. We open 256×512 threads in total, which means that 131072 sets of k consecutive bases can be calculated each time.

However, despite such a large number of threads, it is impossible to process millions of gene sequences at one time. Therefore, we use the grid step technology here: use the for loop in the GPU function and set the step size to the total number of threads in the grid (griddim $\times$ blockdim). This not only solves the problem that the amount of data is larger than the number of threads, but also realizes the reuse of threads, effectively reducing the overhead of CUDA thread startup and destruction.

3 Results

3.1 k-mer

We evaluate the performance of the k-mers model on 20,000 test reads under different measurement, with parameters k = 3-9 through accuracy and time cost (Table 1). While Euclidean distance, Pearson correlation coefficient, and cosine outperform Manhattan distance (Fig 1), the k-mers model underperforms in all respects. As k increases, the accuracy barely improves, while the time cost increases exponentially. The time complexity of this k-mers model is O(maxM, 4^k) because G must be traversed to determine the distance. M is the total number of bases in 10 genomes. A possible reason for the failure of k-mers to complete the assignment may be that the statistical distribution of k-mers reflects the characteristics of the genome, but short local fragments may not contain enough information for the whole genome, so there is a certain gap between global and local fragments.

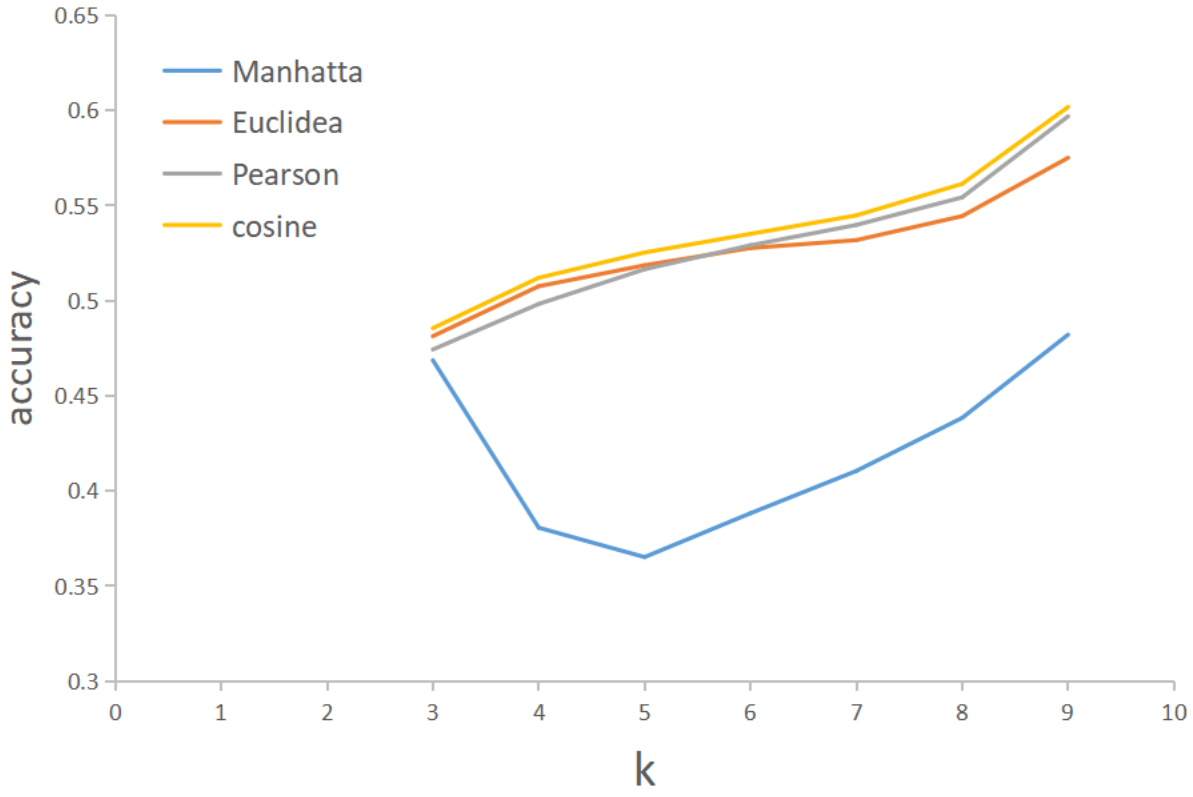

Figure 1: accuracy of different criterion of k-mer model

k	Manhattan Distance		Euclidean Distance		Pearson Correlation Coefficient		Cosine	
	accuracy	CPU time	accuracy	CPU time	accuracy	CPU time	accuracy	CPU time
3	0.47	153.95	0.48	98.73	0.47	107.58	0.49	88.06
4	0.38	227.50	0.51	125.63	0.50	129.17	0.51	113.72
5	0.37	250.06	0.52	159.31	0.52	162.67	0.53	140.39
6	0.39	386.97	0.53	222.38	0.53	246.05	0.54	274.78
7	0.41	2235.27	0.53	3103.13	0.54	481.78	0.54	2562.81
8	0.44	15228.73	0.54	14308.97	0.55	1840.09	0.56	13881.28
9	0.48	51637.45	0.58	50772.11	0.60	60907.69	0.60	45837.34

Table 1: accuracy and time costs(s) of different measurement of test sequences based on k-mer model

k	single direction		double direction	
	accuracy	CPU time	accuracy	CPU time
3	0.604	142.328	0.579	208.906
4	0.636	171.250	0.608	254.984
5	0.663	189.000	0.632	284.438
6	0.696	214.484	0.666	307.094
7	0.779	217.688	0.748	305.156
8	0.915	229.938	0.898	289.703
9	0.989	235.391	0.986	279.422

Table 2: accuracy and time costs(s) of different K in kMM model

3.2 k^{th}-order Markov Model

We assessed k^{th}-order Markov Model's preformance on 20000 test reads, taking parameter k = 3–9, by accuracy and time costs (table 2).Here single and double direction(Calculate both forward and backward, and then assign the sequence to the genome with lowest score) are both implemented. We can see that with the increase of k, the accuracy goes up aggressively, with accuarcy$\approx 98.6\%$ at k = 9. Thus, we can draw the conclusion that k^{th}-order Markov Model performs well. It's also shown in the table 2 that the time cost is stable, for the time complexity of this model is O(M+N), where M is the total base count of 10 genomes and N is total base count of all the query sequence, which are both constants. However, double direction method performs not so good as single direction, which is beyond our expectation. The possible reason is that the direction of the given test sequence is clear because it is simulated data. But it still make sense when we apply the model to real world data.

3.3 GPU acceleration

Based on python cuda, we optimized the steps of calculating statistical matrix in KMM algorithm: calling grid with the size of 2058×512 for GPU calculation to count the value of each element in (4×4^k) matrix. As for K-mer algorithm, we use GPU to optimize the steps of counting k-mer(4^k) vectors, and at the same time optimize the steps in the corresponding functions.

Limited to some functions of python cuda, we can't optimize the GPU in the whole process. However, partial optimization also makes the calculation time greatly reduced. table 3 and 4 show the total time calculated after GPU optimization, with the relative saving time rate (CPU time-GPU time)/(CPU time) attached.

We can speculate from the results that calling GPU itself will take a little more time, so the efficiency improvement is not obvious when the value of k is small. However, with the increase of the value of k and the amount of computation, the advantage of parallel computing efficiency of GPU will become more and more obvious.(Fig 2)

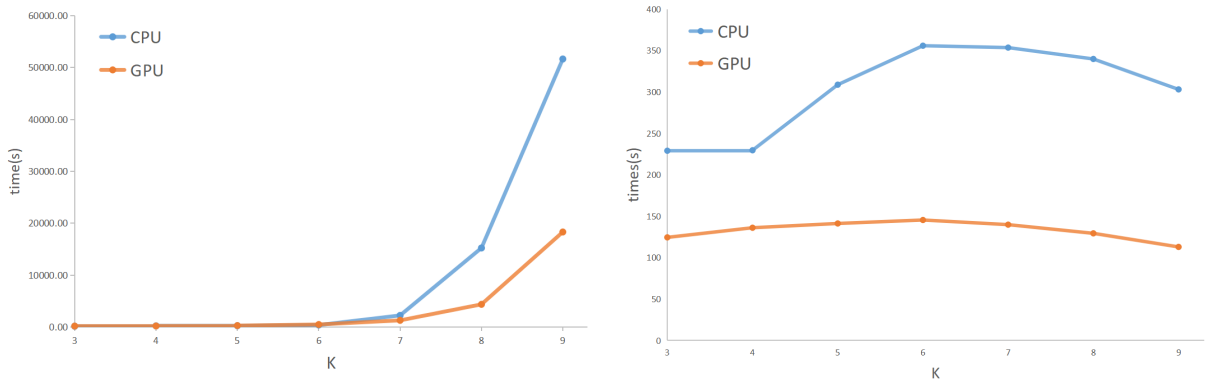

Figure 2: The left hand side plane show the comparison of time consuming between CPU and GPU relative to the k-mer model(here we only use Euclidean criterion); The right hand side plane is the comparison relative to kMM model

4 Question Answer

1.the total number of short sequences in the file reads.fa;

```
assassin@PC-20200423IYUN:/mnt/e/bi371/projet/project1$ grep ">" reads.fa | wc -l
1876
```

We can see the total number of short sequences in the file reads.fa is 1876.

2.(*) the number of reads that can be assigned to a genome in the genome dataset (with whatever criteria you use);

	single direction		double direction	
	GPU time	save time rate	GPU time	save time rate
3	124.422	0.1258	239.610	-0.1422
4	136.047	0.2056	165.031	0.3528
5	141.203	0.2589	174.094	0.3879
6	145.406	0.3221	174.25	0.4326
7	139.781	0.3579	167.453	0.4513
8	129.297	0.4377	133.531	0.5391
9	112.906	0.5203	113.984	0.5921

Table 3: GPU time costs(s) and relative saving time rates of different K in kMM model

	Manhauan Distance		EucHdean Distance	
	GPU time	save time rate	GPU time	save time rate
3	212.125	-0.3779	175.297	-0.7754
4	259.781	-0.1419	199.844	-0.5908
5	309.781	-0.2388	274.375	-0.7222
6	397.625	-0.0275	486.734	-1.1888
7	1084.7656	0.5147	1281.578	0.5870
8	4023	0.7358	4368.0625	0.6947
9	14913.875	0.7112	18303.625	0.6395

Table 4: GPU time costs(s) and relative saving time rates of different K in k-mer model

Minimum number of short sequences in a group	1	5	10	50
Number of groups	10	10	10	8

Table 5: result of question 4

specie	number
Baumannia	295
Psychromonas	20
Sphingomonas	77
Roseiflexus	377
Hydrogenobaculum	100
Alteromonas	276
Denitrovibrio	111
Frankia	16
Candidatus	345
Corynebacterium	259

Table 6: result for question 5

Since both k-mer and kMM models can assign any sequence to the reference genome,we assume all reads can be assigned to a genome in the genome dataset.

3.(*) the number of reads that can't be assigned to any genome in the genome dataset ;

0

4.the number of groups with at least j short sequences assigned, where j = 1, 5, 10 or 50. You can describe the results in a table similar to the one below.

The results are list in table 5.

5. for those groups with at least 10 short sequences assigned, list the total numbers of short sequences assigned to those groups. Then you can draw a pie chart to show the relative frequency of each group.

We list the total numbers of short sequences assigned to those groups as in table 6 and a pie chart (Fig 3) also given to visualize the result.

6. For k=3-9, which k value gives the best classification result? Please give your own criteria for "good" result, and explain why this k value gives the best result.

Since we can directly assess the performance of different k in test reads(the annotation have been given), we can refer to the accuracy and time cost under different k as in table 2. We can see when k=9 we can get the best result.

7. Some researches have shown that combining different orders of Markov Models may improve

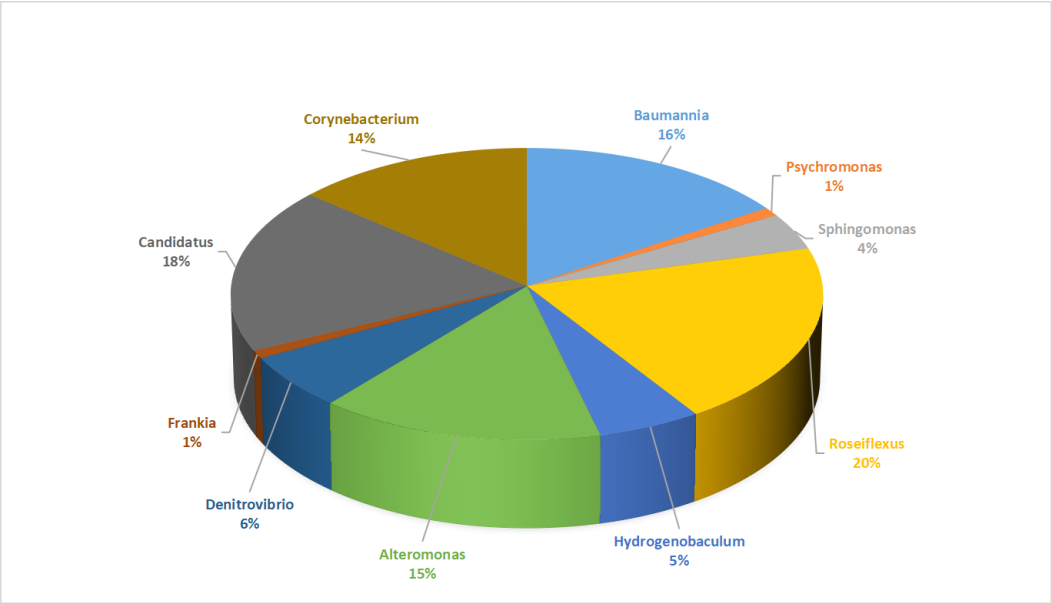

Figure 3: pie chart for question 5

the classification result. Please try different combinations of two or three different k^{th}-order Markov Models, and compare the sequence classification results to that of the best kth -order model to see if there is a combination method performing better than all single-order Markov Model.

Here, we adopt two ways to combine different K order. The first is to add the scores of one sequence under multiple K order directly, and then assign the sequence to the genome with the lowest score. The second way is to first do a softmax transform on the orginal score, and then add up each transformed score and assign the short read to the genome with the lowest score. Here we summary the result in table 7. The result shows when softmax is implemented, performance is lightly better than non-softmax. But since the 9^{th} order Markov Model have reached a surprisingly good performance, the combination method is not necessarily better than it.

combination of different k	non-softmax		softmax	
	accuracy	CPU time	accuracy	CPU time
[5, 7]	0.762	377.094	0.779	436.328
[3, 5]	0.653	342.438	0.663	418.000
[4, 8]	0.903	440.953	0.915	505.563
[8, 9]	0.985	502.953	0.988	563.750
[3, 9]	0.985	403.219	0.989	472.188
[5, 9]	0.985	469.563	0.989	520.750
[7, 9]	0.985	473.391	0.988	471.656
[3, 6, 9]	0.982	530.609	0.989	603.734
[5, 7, 9]	0.981	590.422	0.988	605.750

Table 7: results of combining different orders of Markov Models

References

[1] David Ainsworth et al. "k-SLAM: accurate and ultra-fast taxonomic classification and gene identification for large metagenomic data sets". In: *Nucleic acids research* 45.4 (2017), pp. 1649–1656.

[2] Florent E Angly et al. "Grinder: a versatile amplicon and shotgun sequence simulator". In: *Nucleic acids research* 40.12 (2012), e94–e94.

[3] Rajeev K Azad and Mark Borodovsky. "Effects of choice of DNA sequence model structure on gene identification accuracy". In: *Bioinformatics* 20.7 (2004), pp. 993–1005.

[4] Florian P Breitwieser, DN Baker, and Steven L Salzberg. "KrakenUniq: confident and fast metagenomics classification using unique k-mer counts". In: *Genome biology* 19.1 (2018), pp. 1–10.

[5] David J Burks and Rajeev K Azad. "Higher-order Markov models for metagenomic sequence classification". In: *Bioinformatics* 36.14 (2020), pp. 4130–4136.