Johns Hopkins University

Computational Genomics

EN 600.639

# Augmenting Phylogenetic Classification of Metagenomic Reads

*Authors:*
Shuya Chu,
Stephen Cristiano,
Bing He,
Zhou Ye

*Instructor:*
Dr. Ben Langmead

December 6, 2013

**Abstract**

In recent years, metagenomics has gained increased interest in the scientific community. With relevance to public health, agriculture, biofuels, and marine biology [Wooley et al (2010)], it is important to understand the taxonomic composition of metagenomic samples and its functional impact it their respective ecosystems. In this project, we use simulated data to explore a variety of methods to taxonomically classify metagenomic reads to their correct position in a phylogenetic reference set. In particular, we use three methods: bloom filter, bowtie, interpolated Markov model and we find bloom filter and bowtie have high accuracy for classifying metagenomics reads to our reference genomes.

# 1 Introduction

The recent expansion of next generation sequencing technologies have greatly benefited our ability to study microbial communities. Metagenomics, defined in 2005 as "The application of modern genomics techniques to the study of communities of microbial organisms directly in their natural environments, bypassing the need for isolation and lab cultivation of individual species" [Chen, Pacther (2005)], has seen a surge of research efforts in recent years in diverse fields including public health, agriculture, biofuels, and marine biology, and efforts such as the Human Microbiome Project generating large quantities of metagenomic data. Despite this, computational methods are still needed for analyzing and making sense of these data in an accurate but efficient matter.

One important challenge is, given a set of metagenomic reads, to accurately classify the taxonomic composition of the reads. For example, a metagenomic human gut sample will contain many different microbial species. We wish to be able to classify each read to the correct phylogenetic location in a reference set and determine the presence and abundance of different microbial species in the sample. Adding to the challenge, many of the species will be closely related (belonging to the same genus, family etc) and in a microbial sample, many of the reads make come from *de novo* genomes, *i.e.* from species that have never been sequenced before and unknown to the scientific community.

In this project, we propose, implement, and compare three methods for the classification of metagenomic reads. In particular, we provide a Bloom filter approach, an approach based on Interpolated Markov Models, and an approach based using Bowtie to perform local alignment to reference genomes and classify using a novel algorithm. While similar methods exist, we also consider a number of ways in which we can improve upon these.

To perform our analyses, we obtain reference genomes from RefSeq, a large collection of genomes maintained by the National Center for Biotechnology Institute (NCBI) [Pruitt et al (2009)]

# 2 Related work

A large body of work has already been done on the phylogenetic classification of metagenomic reads. It's important to note that many of the early methods are only accurate for large sequence reads and fail at correctly classifying the short ($\sim$ 100 bp) reads generated by next generation sequencers [Brady, Salzberg (2009)]. We briefly review a few of the more recent metagenomic classification methods.

## 2.1 PhymmBL

PhymmBL [Brady, Salzberg (2009)] uses an Interpolated Markov Model to phylogenetically classify genomic reads to genomes in RefSeq, a large collection of genomes. It is capable of performing classification at different levels of the phylogentic tree (species, genus, family etc). In addition to classification using an IMM, PhymmBL also uses BLAST for local alignment to classify. By applying a linear function of the scores from the IMM and BLAST, the authors report greater classification accuracy than either the IMM or BLAST alone. Though very accurate, PhymmBL suffers from extremely slow running time, due to using BLAST for local alignment.

## 2.2 FACS

Fast Accurate Classification of Sequences (FACS) [Stranneheim et al (2010)] transforms the reference genomes to Bloom filters and query Bloom filters for exact matches. Bloom filters are an compact hash-based data structure which are used to quickly determine whether an element is a member of a set or not. In a trade-off for compactness and speed of look-up, Bloom filters come with a risk of giving false positives which must be controlled. FACS is implemented in PERL and achieves high speed, but is limited to using Bloom filters $< 312MB$ and does not have a scoring system optimized for classification. In a comparison of classification software, other authors have reported an inability to reproduce the results reported in the FACS paper [Bazinet, Cummings (2012)].

## 2.3 Bowtie

Bowtie [Langmead et al (2010)] utilizes a Burrow-Wheeler Transform and an FM index to perform extremely fast alignment of short DNA reads to a reference genome. By performing local alignment of the reads to each of the reference genomes from a phylogenetic tree, and using an appropriate scoring system, Bowtie can be adapted for classification of metagenomic reads.

# 3 Methods and software

We obtain our reference genomes from RefSeq, obtained from the NCBI. Due to the vast size of RefSeq, we initially considered only the phylum *Proteobacterium*, but even that exceeded our memory limits. For this reason, we decided to use comparisons of the methods for our reference set the genomes from *Escherichia coli* (strain K-12 substain. DH10B) and *Bdellovibrio bacteriovorus*. To simulate our data, we use the software MetaSim [Richter et al (2008)] which will generate metagenomic reads into a FASTA file given a set of reference genomes.

As the main purpose of the Bowtie method was to develop an algorithm that will not have align to every genome in a reference set, we also considered 15 genomes from *Proteobacterium* but the same simulated reads. However, the comparisons with the other methods were still conducted only on the two previously mentioned reference genomes.

## 3.1 Bloom Filter augmented Classification (BFAC)

In this section, we describe the Bloom Filter augmented Classification (BFAC) method. In a nutshell, BFAC is a bloom filter based classification methods for metagenomic phylogenetic classification. Currently the only relevant work we can find is the FACS methods [Stranneheim et al (2010)]. FACS methods can classify metagenomic sequences as belonging or not belonging to a reference sequence. To fulfill this goal, FACS counted the matched k-mers of a query sequence and set a threshold to make a dichotomized classification. Our BFAC method augmented FACS to be applicable in the situation of metagenomic phylogenetic classification. There are two major augmentations. First, in addition to dichotomized classification scheme as implemented in FACS, BFAC can not only make multi-category classification but also perform phylogenetic classification. More specifically, when species-level classification fails for a query sequence, BFAC will automatically classify the sequence to higher level (e.g., genus level) based on an entropy-based summary statistics based on the species-level scores. Second, BFAC speeds up the querying process by a two-step scan system: a coarse scanning and a detailed scanning.

In BFAC, we first build a bloom filter for the reference genomes of species in our selected subtree of the NCBI taxonomy. Each reference genome is decomposed into k-mers with a specified length; those k-mers are then inserted into the initialized Bloom Filter as keys. In order to fully take advantage of the Bloom Filter data structure, we initialize a differently-sized bloom filter for different reference genome based on genome sizes and total number of k-mers. When initializing the bloom filter, we also specify the acceptable false positive rate. As it is shown in previous study [Broder, Mitzenmacher (2004)], the size of the bitmap grows linearly with the number of elements in the bitmap. Note that we explore how the specified k-mer length and false positive rate would affect our classification accuracy and time used to query (please refer to relevant figures in the result section). Second, once the filters are built, we can interrogate them with multiple metagenomic reads and record the match scores. More specifically, metagenomic reads are decomposed into k-mers with

the same length and are used ask keys for querying bloom filters. Third, we perform the phylogenetic classification of each reads and output the results. We implemented the BFAC in perl using perl module BLOOM::FASTER based on the FACS code.

There are two things we tried for which we didnt present results here. The first thing is to incorporate false positive rate of bloom filter into the scoring system. We penalized the scores according to the uncertainty implied by the false positive rate of the bloom filter. We didnt present the results because we found that the classification accuracy change very little when we use different false positive rate (from 0.0005 to 0.0035, please refer to figures in results section). The second thing we tried is to use a variant of classic bloom filter for our BFAC. We explored scalable bloom filters that can dynamically adapt to the stored keys while keeping an acceptable false positive rate; and the attenuated bloom filters with a depth large than 1 to store the set of species under the same genus instead of building a bloom filter for each reference genome. We came across some technical problems and didnt have enough time to complete, so we did not present the results here.

## 3.2 Improvement with Bowtie2

Bowtie2 is an ultrafast aligner using FM index. It is particularly good at aligning reads of about 50 up to 100s or 1,000s of characters. Since our simulate testing data is around 100-200 long, using Bowtie2 will lead to fast aligning.

The idea for improvement is by skipping some unnecessary alignment between the test data and reference, identifying the species for test data will be much more fast. Like what we did in the Boyer-Moore method in class, we use results from comparisons to skip future alignments that definitely won't match. To be more specific, if we got a really low score when aligning the test data with one species, it means the sample is definitely not from this species, thus we can skip the whole species from the same genus. For example, if we know this sample definitely does not belong to *Canis*, instead of checking whether it's *Canis lupus*, we can check whether it's *Cuon alpinus* from the genus of *Cuon*.

First we construct a tree structure to describe the taxonomy relation between the reference in python. For each node, node.data represent directory path of this reference, parent and children is directory path for its parent and children in the reference subtree we used.

Then by running Bowtie2, we will have scores which describe the similarity between the sample data and reference. If scores is really low, in other words, we have enough confidence to say the sample is not from this species, we skip the whole genus this reference species belongs to. If scores is not that bad, we will go through all the species in this genus to determine which species it has most similarity with.

Threshold here is very important. We didn't come up with a fixed expression to calculate it by the due date. We extract 31 species from 18 genus from the reference set and set the threshold after observing the result from those alignments. Also, we used the average score from the scores of 1000 test sequence in the .fna file.

In order to run the code, please change the value of 'projectpath' to the actual path that

'Bowtie.tar.gz' you extract to. And also, make sure inside the 'projectpath', there is a /data/index folder and a /bowtie2-2.1.0 folder.

### 3.3 Interpolated Markov Model

DNA sequence can be represented by a sequence of random variables $X_1, X_2, \cdots, X_n$ where $X_i$ is the base at position $i$ which can take a value from $(A, T, C, G)$. Markov model assumes the probability of $X_i$ taking on some value will depend on $X_{i-1}, X_{i-2}, \cdots, X_{i-n}$. $n$ is the number of order which shows how many bases we need to predict $X_i$. In this way, any DNA sequence can be scored by computing the probability for different position generated by Markov model.

There is a trade-off in Markov model. For an n-th order Markov model, the parameters to estimate on one position for a fixed-length DNA sequence is $4^{n+1}$. If we use low-order Markov model, there are fewer parameters but less prediction power. However, if we use high-order Markov model, we will obtain strong prediction power but the parameters will be exponentially large. It is very likely that there are insufficient data to learn the parameters.

Interpolated Markov model (IMM) successfully combines the merits of low-order and high-order models by using a linear combination of Markov models. The idea is generally from Salzberg's paper[Salzberg et al (1998)]. All the mathematics are in the appendix and we think the threshold in the original paper for $\xi^2$-test 0.5 is incorrect. The confidence score i.e. the p-value should be smaller than 0.05. Correct us if you think we are not right.

We implement IMM all by ourselves using Java and the speed is very good. We have tried our best to minimize the times of scanning data set. I also do something Salzberg did not do in his paper, which is show the effect of order and threshold.

The general steps we follow to implement IMM are:

1. Training the parameters of IMM from reference genomes;
2. Use the parameters of each reference genome to score the test genomes;
3. Label the test genomes with the reference genomes which have highest score;

Here we do something a little different. After getting the score, we use a second threshold to screen the score. We call it score standard. When the score is too low (below threshold), we treat the test genome unclassified; otherwise, we use the same tool to label the test genome.

## 4 Results

### 4.1 Evaluation Method

We compute five statistics to do the comparison in this paper. They are classification accuracy, sensitivity of B, specificity of B, sensitivity of E and specificity of E where B is

| | E | B |
|---|---|---|
| E | EE | EB |
| B | BE | BB |
| N | NE | NB |

| Method | Time |
|---|---|
| Skip | 6.3978 |
| No Skip | 12.2367 |

Table 1: Bowtie Comparison

Bdellovibrio bacteriovorus and E is Escherichia Coli. We use "B" and "E" to notate two bacteria for the entire paper. The formula to compute them are as following:

where $EE, EB, BE, BB, NE, NB$ are counts. Here $N$ means we neither classify the test genome B not E.

$$
\begin{aligned}
Acc &= \frac{EE + BB}{EE + EB + BE + BB + NE + NB} \\
Sen(B) &= \frac{BB}{BB + EB + NB} \\
Spe(B) &= \frac{NE + EE}{EE + NE + BE} \\
Sen(B) &= \frac{EE}{EE + BE + NE} \\
Spe(B) &= \frac{BB + NB}{EB + BB + NB}
\end{aligned}
\tag{1}
$$

| Order | Accuracy | Sens(B) | Spec(B) | Sens(E) | Spec(E) |
|---|---|---|---|---|---|
| 2 | 0.4781 | 0.023 | 0.943 | 0.9332 | 0.026 |
| 4 | 0.4626 | 0.1816 | 0.7478 | 0.7436 | 0.1826 |
| 6 | 0.4975 | 0.0014 | 0.9966 | 0.9936 | 0.0018 |
| 8 | 0.3341 | 0 | 1 | 0.6682 | 0.1912 |
| 10 | 0.3341 | 0 | 1 | 0.6682 | 0.1912 |

Table 2: Comparison For Different Orders

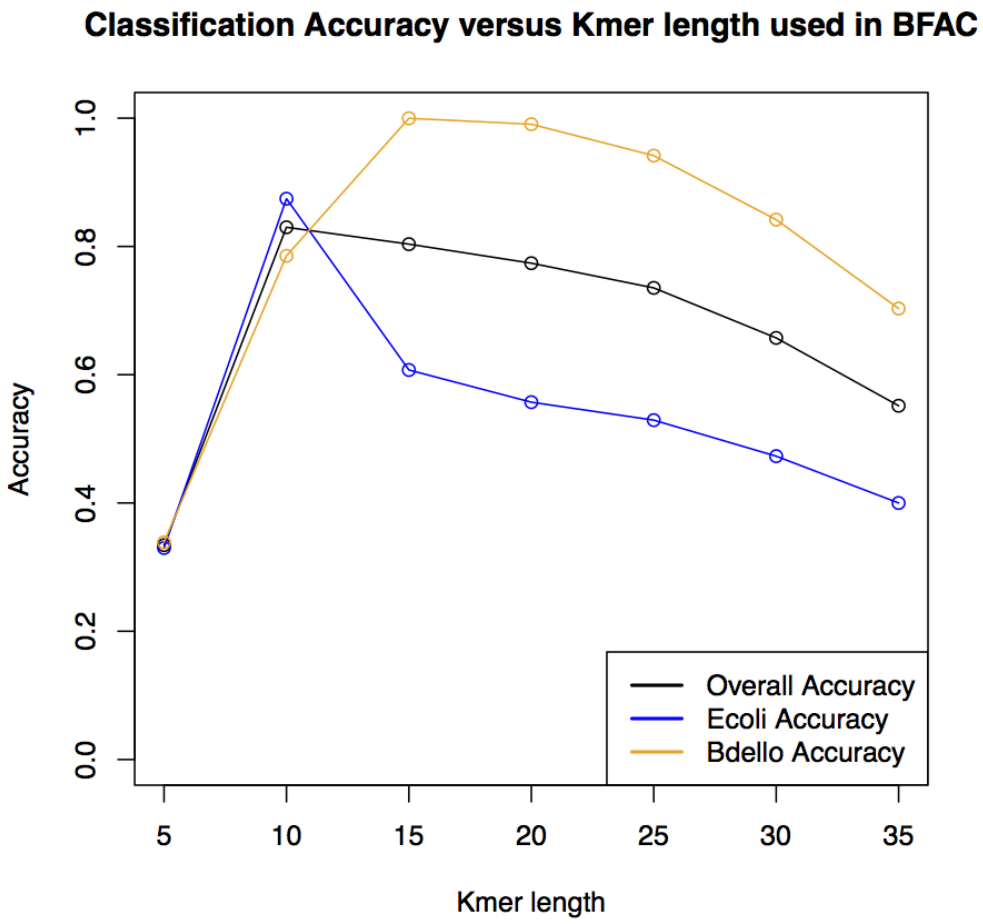


Figure 1: Classification accuracy versus Kmer length used in BFAC

| Threshold | Accuracy | Sens(B) | Spec(B) | Sens(E) | Spec(E) |
|---|---|---|---|---|---|
| 200 | 0.2054 | 0 | 1 | 0.4108 | 0.4338 |
| 300 | 0 | 0 | 1 | 0 | 0.9998 |
| 400 | 0.4975 | 0.0014 | 0.9966 | 0.9936 | 0.0018 |
| 500 | 0.0204 | 4.00E-04 | 0.9994 | 0.0404 | 0.9296 |
| 600 | 0.2087 | 0.0066 | 0.9838 | 0.4108 | 0.4338 |

Table 3: Comparison For Different Thresholds

| Standard | Accuracy | Sens(B) | Spec(B) | Sens(E) | Spec(E) |
|---|---|---|---|---|---|
| 50 | 0.4977 | 0.0018 | 0.9936 | 0.9936 | 0.0018 |
| 60 | 0.4975 | 0.0014 | 0.9966 | 0.9936 | 0.0018 |
| 70 | 0.233 | 0 | 1 | 0.466 | 0.365 |
| 80 | 0.0042 | 0 | 1 | 0.0084 | 0.9826 |
| 90 | 0 | 1 | 1 | 0 | 0.9996 |

Table 4: Comparison For Different Score Standards

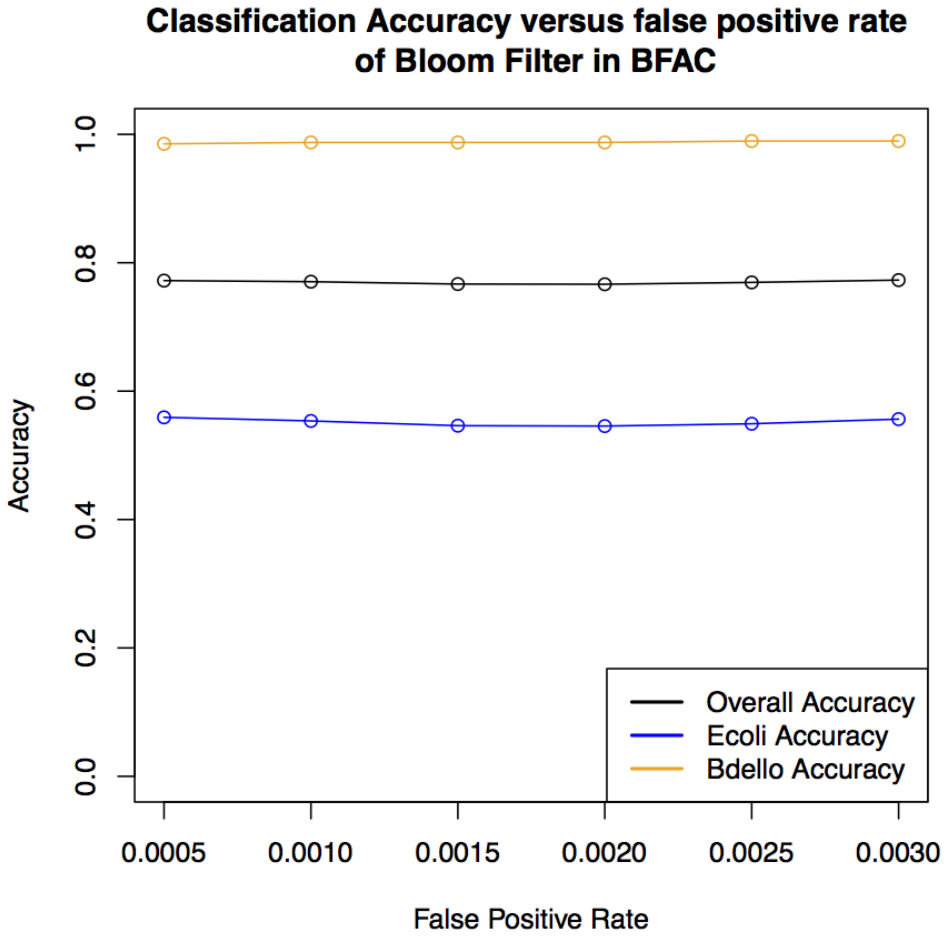


Figure 2: Classification accuracy versus false positive rate of bloom filter in BFAC

| Method | Accuracy | Sens(B) | Spes(B) | Sens(E) | Spes(E) | Time(s) |
|---|---|---|---|---|---|---|
| Bowtie | 0.9649 | 0.9298 | 1 | 1 | 0.9702 | 12 |
| Bloom Filter | 0.8036 | 0.6074 | 0.5 | 0.9998 | 0.5 | 52 |
| IMM | 0.4975 | 0.0014 | 0.9966 | 0.9936 | 0.0018 | 14 |

Table 5: Comparison of All Methods

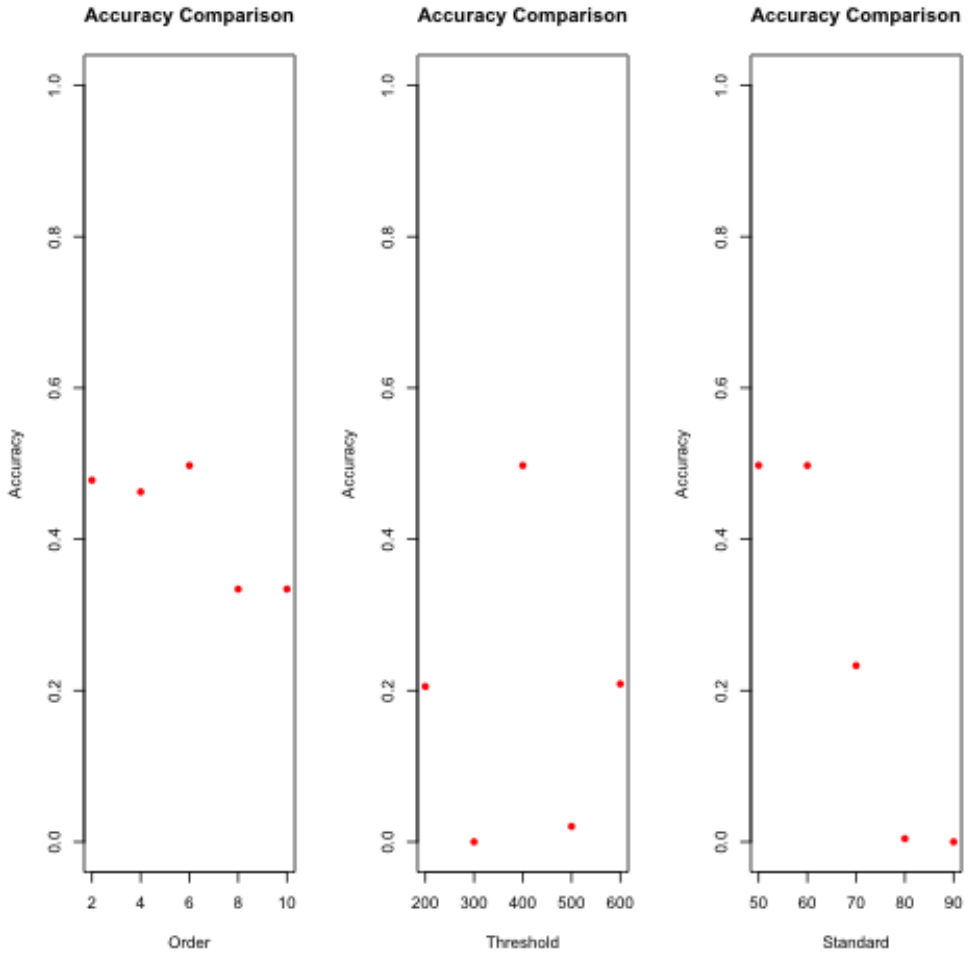


Figure 3: All Comparison

### 4.2 All Results

# 5 Discussion

## 5.1 Bloom Filter augmented Classification (BFAC)

The figure 1 shows how the classification accuracies change in response to the specified Kmer length when build the bloom filter. As we can see from the figure, the classification accuracy for Bdello and Ecoli as well as the overall accuracy are quite low when the kmer length is 5. This is reasonable as short kmers can occur frequently in different reference genomes as they are not specific enough for capturing the source of metagenomic reads. The accuracies increase as the kmer length increase to around 15 bp and decrease as the kmer length further increase to 35. This is probably due to the fact that longer kmer is more likely to contain errors and become harder to get classified. Based on the figure the optimal length of kmer to be used is between 10-15 bp. We think that this number is generally applicable to bacteria. Note that the time for building bloom filter and the bitmap sizes also change in response to the length of k-mers used.

The figure 2 presents how the classification accuracies change with regard to false positive rate used to build bloom filter in BFAC. Surprisingly, the Bdello and Ecoli accuracy as well as the overall accuracy vary very little over different values of false positive rate. It is probable that we increase the false positive rate more aggressively, the accuracy will decrease.

## 5.2 Bowtie

After running `bowtie_with_skipping.py` and `without_skipping`, we can tell with 31 species in 18 genus, skipping method is 2 times faster. Since we didn't make the index for the whole subtree we used, the improvement is not very obvirous. The larger the reference set it is the more improvment we can observe.

## 5.3 IMM

To compare the effect of orders, I use five different choices of orders $\{2, 4, 6, 8, 10\}$. The results are in Table 2 and Figure 3 (left). We can see that order 6 has the highest accuracy, which is different from the original paper.

In order to compare the effect of different thresholds, I implement five distinct thresholds $\{200, 300, 400, 500, 600\}$ and the results are in Table 3 and Figure 3 (middle). It is obvious to see that 400 has highest accuracy and it is consistent with Salzberg's paper.

Besides, I also want to check if my improvement actually works. I check five standards to threshold the score $\{50, 60, 70, 80, 90\}$. The results are in Table 4 and Figure 3 (right).

Unfortunately, my idea does not work since 50 is not a useful threshold (all scores are larger than 60). It is better to keep the score as it is.

### 5.4 Comparison of Three Methods

As it is shown in table 5, Bowtie yields the highest classification accuracy with highest speed. Bloom filter is slower than bowtie and produces slightly lower accuracy with much smaller storage. IMM models give the lowest accuracy but has relatively high time efficiency.

## 6 Conclusion

In our project, we implemented and compared three methods for the phylogenetic classification of metagenomic reads. Under our simulated data,

For future work, we would ideally run our methods on reference genomes from the full RefSeq and using simulated reads from a larger clade in the tree. While we aimed to do so for our project, we found that we found that the limitations on cluster we were using were too stringent and we had to vastly scale down the scale of our reference set. In particular, we believe that the IMM method will improve as we scale up the data and the number of reference genomes. We also would like to scale our comparisons to real data instead of simulated reads, especially a dataset that has been well studied and we have some idea of what the true classifications should be. It's also important to compare our methods to the existing methods. While we believe we sufficiently improved upon existing methods, we were unable to get existing software running on the cluster, aside from FACS.

Other things we would like to test is the ability of our methods to detect *de novo* reads. A real metagenomics sample will have reads from many bacteria which have never been discovered, which we would like our methods to classify to the correct genus, but identify as being from an unknown genome. One easy way we can explore this is by simulating reads from a reference set of genomes, and then classify against the reference set multiple times using a cross validation procedure with one reference genome left out from each classification.

In summary, we identify three methods for phylogenetic classification for metagenomics. While our initial results look very promising, we need to scale up the size of our simulations and set of reference genomes to fully quantify the accuracy, sensitivity, and specificity of these methods. We remain hopeful that our methods will perform well on a more realistic simulation and can be applied to real metagenomics data.

## Contributions

All group members contributed equally.

We discussed methods and analyses together is a team, with each of us taking a particular coding method. Shuya wrote the bowtie2 method; Zhou wrote the interpolated Markov model; Bing and Stephen wrote bloom filter. We write the report together and also do the presentation together.

# 7 References


Chen K, Pachter L (2005). Bioinformatics for Whole-Genome Shotgun Sequencing of Microbial Communities. *PLoS Comput Biol*, 1(2): e24. doi:10.1371/journal.pcbi.0010024

Brady A, Salzberg S (2009). Phymm and PhymmBL: metagenomic phylogenetic classification with interpolated Markov models. *Nature Methods* 6, 673 - 676 (2009)

Pruitt KD, Tatusova T, Klimke W, Maglott DR. NCBI Reference Sequences: current status, policy and new initiatives. *Nucleic Acids Res.* 2009 Jan; 37 (Database issue):D32-36.

Bazinet AL, Cummings MP (2012) A comparative evaluation of sequence classification programs *BMC Bioinformatics* 2012, 13:92

Stranneheim H, Kller M, Allander T, Andersson B, Arvestad L, Lundeberg J Classification of DNA sequences using Bloom filters *Bioinformatics* (2010) 26 (13): 1595-1600.

Langmead B, Trapnell C, Pop M, Salzberg S Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology* (2010) 10:R25

Richter DC, Ott F, Auch AF, Schmid R, Huson DH MetaSimA Sequencing Simulator for Genomics and Metagenomics. *PLoS ONE* (2008) 3(10): e3373. doi:10.1371/journal.pone.0003373

Wooley JC, Godzik A, Friedberg I A Primer on Metagenomics *PLoS Computatoinal Biology* (2010) doi.1371/journal.pcbi.1000667

Salzberg S, Delcher AL, Kasif S, White O Microbial gene identification using interpolated Markov models. *Nucleic Acids Research* 1998, Vol.26 No.2

Broder,A. and Mitzenmacher,M. Network applications of Bloom lters: a survey. *Internet Mathemathics* (2004), 1, 485509. Down,T.A. et al. (2008)

# Appendix

## 7.1 Interpolated Markov Model

The basic IMM is as following:

$$\begin{aligned} &P_{IMM}(X_i|X_{i-1},\cdots,X_{i-n}) \\ &= \lambda_0 P(X_i) + \lambda_1 P(X_i|X_{i-1}) + \cdots + \lambda_n P(X_i|X_{i-1},\cdots,X_{i-n}) \end{aligned} \tag{2}$$

where $\sum_i \lambda_i = 1$.

We can also represent model in an recursive way:

$$\begin{aligned} &P_{IMM,n}(X_i|X_{i-1},\cdots,X_{i-n}) \\ &= \lambda_n(X_i) P(X_i|X_{i-1},\cdots,X_{i-n}) \\ &+ (1-\lambda_n(X_i)) P_{IMM,n-1}(X_i|X_{i-1},\cdots,X_{i-n+1}) \end{aligned} \tag{3}$$

where $\lambda_n(X_i) \in [0,1]$.

IMM uses weights to guarantee that we use high-order models whenever we have sufficient data and also use lower-order models, where more data is available, to smooth the prediction of high-order models.

## 7.2 IMM Parameter Learning

There are two groups of parameters we need to learn for IMM. One is the distribution of $P(X_i|X_{i-1},\cdots,X_{i-n})$ where $X_i \in (A,T,C,G)$ is the base at position $i$. The other is $\lambda_n(X_i)$ where $n$ is the order of the current Markov model.

We estimate $P(X_i|X_{i-1},\cdots,X_{i-n})$ by counting. The formula to obtain $P(X_i|X_{i-1},\cdots,X_{i-n})$ is:

$$P(X_i|X_{i-1},\cdots,X_{i-n}) = \frac{f(X_i,X_{i-1},\cdots,X_{i-n})}{f(X_{i-1},\cdots,X_{i-n})} \tag{4}$$

where $f$ is the number of occurrences of input string.

The $\lambda$ value associated with $P(X_i|X_{i-1},\cdots,X_{i-n})$ can be treated as a measurement of our confidence in the n-th order Markov model. How can we decide $\lambda_n(X_i)$? When we have sufficient training data (when the number of training data $c$ exceeds some threshold), the high-order model will be preferred. If the training data is insufficient, what we do is to use a $\xi^2$-test to compare the distribution of $(A,T,C,G)$. From the test, we will obtain a p-value $d$. When $d \leq 0.05$, we set $\lambda$ as 0; otherwise, we set $\lambda$ as $\frac{c}{threshold} \times d$. Using this method, when high-order model does not have sufficient data and it has a different distribution at position $i$, we will reject using the high-order model. The way to compute $\lambda$ is as following:

$$\lambda_n(x_i) = \begin{cases} 1.0 & \text{if } c > threshold \\ \frac{c}{400} \times d & \text{if } d \geq 0.05, c \leq threshold \\ 0.0 & \text{if } d < 0.05, c \leq threshold \end{cases} \tag{5}$$