Continuous embeddings for large-scale machine learning with DNA sequences

Romain Menegaux

PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL
Préparée à MINES ParisTech

Continuous embeddings for large-scale machine learning with DNA sequences

Représentations pour l'apprentissage statistique à grande échelle en génomique

Soutenue par
Romain MENEGAUX
Le 7 mai 2021

Ecole doctorale n° 621
Ingénierie des Systèmes, Matériaux, Mécanique, Energétique

Spécialité
Bio-informatique

Composition du jury :

Gregory, KUCHEROV Université Gustave Eiffel	*Président*
Jean-Daniel, ZUCKER Institut de Recherche pour le Développement	*Rapporteur*
Nicola, SEGATA University of Trento	*Rapporteur*
Sophie, SCHBATH INRAE	*Examinateur*
Flora, JAY Laboratoire de Recherche en Informatique	*Examinateur*
Pierre, MAHE bioMérieux	*Examinateur*
Armand, JOULIN Facebook A.I.	*Examinateur*
Jean-Philippe, VERT Google Brain	*Directeur de thèse*

MINES ParisTech | PSL

Abstract

The cost of DNA sequencing has been divided by 100,000 in the past 15 years. Brought along by this technological revolution, ever larger volumes of data are coming in from diverse fields and problems, raising new computational challenges. How can we efficiently store and analyze DNA sequences? A domain that has greatly benefited from this is the one of metagenomics, which seeks to characterize and identify microbes – bacteria, viruses – by sequencing and analyzing their DNA. A modern DNA sequencing experiment outputs billions of short DNA fragments (reads), in random order. A crucial step in the bioinformatics analysis pipeline is to match those fragments to their parent genomes, a problem called taxonomic binning. Up until a few years ago alignment-based strategies were the norm, which were largely based on string-matching algorithms. However these have become too slow for the ever-growing amount of available sequenced genomes, and more recently so-called pseudo-alignment strategies have become standard. These hold databases of large sub-strings and look for matches in the query sequences. Machine learning methods have shown promising success in classifying biological sequences and in this thesis we will investigate these methods for taxonomic binning. Firstly, we present an algorithm, `fastDNA`, that embeds sequences in a continuous vector space by first splitting them into short k-mers (substrings of length k) and learning an embedding for each k-mer. The embedding is then run through a linear classifier. In the second part of this thesis we will present `Brume`, an extension to `fastDNA` that partitions k-mers into clusters based on the de Bruijn graph, and learns one representation per cluster. This allows to increase k and the effective number of k-mers, without needing additional memory. Finally we will introduce Phylo-HS, a structured loss for machine learning based on the phylogenetic tree.

Résumé

Le coût du séquençage de l'ADN a été divisé par 100 000 en seulement 15 ans. Grâce à cette révolution technologique, des volumes de données de plus en plus grands arrivent de domaines variés, posant de nouvelles problématiques informatiques. Comment analyser et stocker les séquences d'ADN de manière efficiente ? Un domaine ayant grandement bénéficié de cette avancée est la métagénomique, qui cherche à caractériser et identifier les microbes – bactéries, virus – en séquençant puis analysant leur ADN. Or le résultat d'une expérience de séquençage se compte en milliards de petits fragments d'ADN (reads), mélangés aléatoirement. Une étape cruciale en bioinformatique est d'identifier le génome d'origine de chacun de ces fragments, problème dit du taxonomic binning. Jusqu'à récemment, les méthodes étaient basées sur l'alignement des séquences à des génomes de référence. Le nombre de ces génomes augmentant, ces méthodes d'alignement sont devenues trop lentes et ont laissé place à un nouveau standard : le pseudo-alignement. Celui-ci consiste à chercher des sous-séquences du read dans une base de données constituée au préalable. L'apprentissage statistique a aussi des résultats prometteurs pour la classification de séquences biologiques, et dans cette thèse nous approfondirons ces méthodes pour le taxonomic binning. En premier lieu nous présenterons un algorithme, `fastDNA`, qui apprend des représentations continues pour tous les k-mers (des courtes sous-séquences de longueur k, les "mots" de l'ADN). On obtient ainsi une représentation vectorielle pour un read en combinant les représentations de ses k-mers. Un classifieur linéaire prédit ensuite la classe du read à partir de ce vecteur. En deuxième partie nous présenterons `Brume`, une extension de `fastDNA` qui partitionne les k-mers en groupes à partir du graphe de de Bruijn, et apprend une représentation par groupe. Ceci permet d'accroître k et donc le nombre de k-mers effectifs sans mémoire supplémentaire. Enfin nous introduirons Phylo-HS, une nouvelle fonction objectif pour l'apprentissage statistique basée sur l'arbre phylogénétique.

“ *Theory is when you know everything but nothing works. Practice is when everything works but no one knows why. In our lab, theory and practice are combined : nothing works and no one knows why.* ”

---

attributed to Albert Einstein

Contents

Abstract i

Résumé iii

Contents vii

1 Introduction 1
- 1.1 Context . . . 3
- 1.2 Taxonomic Binning . . . 9
- 1.3 Supervised Learning Framework . . . 12
- 1.4 Supervised learning for biological sequences . . . 16
- 1.5 Contributions . . . 23

2 fastDNA 29
- 2.1 Introduction . . . 31
- 2.2 Method . . . 32
- 2.3 Experiments . . . 34
- 2.4 Conclusion . . . 40

3 Brume: Embedding the de Bruijn graph 43
- 3.1 Introduction . . . 46
- 3.2 Approach . . . 47
- 3.3 Methods . . . 52
- 3.4 Results . . . 52
- 3.5 Discussion . . . 57
- 3.6 Conclusion . . . 58

4 Adapting the hierarchical softmax for taxonomic classification 61
- 4.1 Introduction . . . 63
- 4.2 Related Work . . . 64
- 4.3 Methods . . . 66
- 4.4 Results . . . 68
- 4.5 Discussion and future work . . . 71

5 Conclusion 73

Bibliography 77

Chapter 1

Introduction

“ You need to let the little things that would ordinarily bore you suddenly thrill you. ”

Andy Warhol

Contents

1.1 Context . . . 3
1.1.1 A quick dive into the microscopic world . . . 3
1.1.2 Metagenomics - revealing microbes through their DNA . . . 3
1.1.3 DNA - the universal biological code . . . 4
1.1.4 Whole genome sequencing . . . 6
1.2 Taxonomic Binning . . . 9
1.2.1 Presentation . . . 9
1.2.2 Alignment-based methods . . . 10
1.2.3 Pseudoalignment . . . 11
1.2.4 DNA-to-protein . . . 11
1.2.5 Binner Evaluation . . . 12
1.3 Supervised Learning Framework . . . 12
1.4 Supervised learning for biological sequences . . . 16
1.4.1 Range of applications . . . 16
1.4.2 Natural Language Processing . . . 17
1.4.3 Bag-of-words representation . . . 17
1.4.4 Word embeddings . . . 18
1.4.5 Convolutional Neural Networks . . . 19
1.4.6 Recurrent Networks . . . 20
1.4.7 Transformers . . . 20
1.4.8 End to end learning . . . 22
1.5 Contributions . . . 23
1.5.1 fastDNA . . . 23
1.5.2 Brume . . . 24
1.5.3 Phylo-HS, a structured loss for taxonomic classification . . . 26
1.5.4 Published work appearing in this thesis . . . 28

Abstract

This chapter sets the context for the contributions presented in this thesis. In the first biology-oriented part we will give an overview of DNA sequencing technology and its applications, notably to the field of metagenomics: the study of microbes from their genetic material. We will also briefly describe the bioinformatics challenges that have arisen to process the enormous quantities of data output by DNA sequencing experiments. In the second part we will focus on one of those challenges: how to assign DNA fragments back to their genome of origin – the so-called taxonomic binning problem. We will review the state-of-the-art bioinformatics methods to approach it. In the third part we will present the applications of statistical learning – and deep learning in particular – to genomics. Finally we will end the chapter by presenting our contributions.

Résumé

Ce chapitre pose le contexte des contributions présentées dans cette thèse. Dans la première partie, orientée biologie, nous donnons une vue d'ensemble des technologies de séquençage d'ADN et de leurs applications, notamment au domaine de la métagénomique : l'étude des microbes à partir de leur matériel génétique. Nous décrirons aussi brièvement les défis bioinformatiques posés par les quantités énormes de données issues des expériences de séquençage. Dans la deuxième partie nous nous concentrons sur l'un de ces défis : comment retrouver le génome d'origine d'un fragment d'ADN – problème dit du *taxonomic binning*. Nous passerons en revue l'état de l'art des méthodes bioinformatiques utilisées pour y répondre. Dans la troisième partie nous présentons les applications de l'apprentissage statistique – et notamment de l'apprentissage profond – à la génomique. Nous terminons ce chapitre par une présentation succincte de nos contributions.

1.1 Context

1.1.1 A quick dive into the microscopic world

Although invisible to the human eye, microorganisms – bacteria, viruses, fungi – are everywhere and play a fundamental role in all ecosystems. Understanding them and their communities has both industrial and scientific applications in ecology, energy and medicine. Although we first come to know of them as the agents of infectious diseases (then labelled as pathogens), they can also help to cure or fight them: antibiotics are produced by industrial microbiology [Gupta et al., 2020], dengue-carrying mosquitoes have been sterilized with bacteria [Caragata and Walker, 2012]. Since the 1980s, bacteria have been harnessed by bio-engineers to mass-produce valuable proteins such as insulin [Baeshen et al., 2014], vitamins or enzymes [Vandamme, 1992]. Microbial cells can also serve as energy sources – then named Microbial Fuel Cells (MFCs)[Scott and Murano, 2007] – recycling factories [Goglio et al., 2019] or carbon sinks. Even if the thought of mankind forcing these billions of tiny living creatures into slavery appalls you, simply observing them can yield crucial information. Indeed, microbial communities can function as trackers or indicators of an environment's, or of our own, changes in health.

There are approximately 10 times more bacterial cells than human cells in our own body, spread in different communities. The largest of these, the gut microbiome, has been the focus of increased research in the past 20 years. Not only essential in digesting our food intake, its composition has been linked to a growing variety of health conditions such as bowel diseases, mental health, or obesity [Fan and Pedersen, 2021]. With massive research endeavors such as the Human Microbiome Project ([Huttenhower et al., 2012]), new roles and functions are being discovered every year, enough for it to be considered as an organ of the human body in its own right [Zimmer, 2010].

Although we are now convinced of their importance, there is still so much about them that we do not know. How then can we characterize microbial communities and the specific roles they play?

1.1.2 Metagenomics - revealing microbes through their DNA

Biologists have been classifying and organizing living organisms for hundreds of years, and methods have evolved accordingly to the knowledge and technologies available. *Taxonomy* is the field of identifying and naming organisms, while *phylogeny* is the task of organizing them in virtue of their evolutionary relationships. Before the advent of DNA sequencing, both of those tasks were based on similarities or differences in *phenotype* – observable traits – such as size, color or regime. Until well into the 20th century, microbial communities were studied by isolating samples of bacteria then cultivating them in a laboratory environment, until they were in sufficient number for their physiological and cellular characteristics to be studied. Aside from being time-consuming and error-prone, this process suffered from a major setback: the counts of cells obtained in cultivation were systematically lower than those observed in the environment. This is because a majority (an estimated $> 99\%$, [Hugenholtz et al., 1998]) of microbial species are *unculturable*: they cannot be cultivated in current laboratory settings.

This problem was largely solved by identifying microbes through their most intimate information: their DNA. Not only does knowing an organism's DNA enable identifying mutations, tagging and distinguishing species or classes of cells, it also has the advantage of lingering around

the environment, revealing otherwise unobservable species. The budding field of *metagenomics* – "the application of modern genomics techniques to the study of communities of microbial organisms directly in their natural environments" [Chen and Pachter, 2005] – arose as a way to circumvent the limitations of lab cultivation, and has emerged as the method of choice for studying microbial communities [Qin et al., 2010]. In fact now one could say the relationship has been reversed: new unknown organisms are classified with regards to their genotype rather than their phenotype [Hug et al., 2016]; [Parks et al., 2018].

The range of sub-fields of metagenomics can be broadly broken into three categories, which we define below along with some of their applications:

(i) *Species detection*: pathogen detection for infectious disease diagnosis.

(ii) *Abundance profiling*: estimating the respective proportions of organisms. This can serve for disease diagnosis or as a biological indicator.

(iii) *Functional profiling*: measuring the gene expression/protein-levels/metabolic activities (see section 1.1.3) to understand the communities' capacities and the roles it plays in its environment.

In this work we will mostly focus on the first two problems (i) and (ii), which we will describe in more detail after a brief overview of DNA.

1.1.3 DNA - the universal biological code

Deoxyribonucleic acid (DNA) is the fundamental building block of all life on Earth, coding the necessary information for an organism to grow and react to its environment. Stored in each cell of an organism, DNA is a macro-molecule composed of two chains (*strands*) of nucleotides: Adenine (`A`), Cytosine (`C`), Guanine (`G`) and Thymine (`T`). Each strand is the "mirror image", or *reverse complement* (RC) of the other, as nucleotides are chemically bound to their mirror: `A` with `T` and `C` with `G` (illustrated in the upper part of figure 1.1). Due to this, a nucleotide is also called *base pair (bp)*, justifying the following representation:

Mathematical notation:

One needs only one strand to represent all the information coded by the DNA. Conceptually, a DNA sequence is therefore represented as chain (or *string*) of L characters $\mathbf{x} = \mathrm{x}_1 ... \mathrm{x}_L$ from a four-letter alphabet $\mathcal{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Its reverse-complement is $\bar{\mathbf{x}} = \overline{\mathrm{x}_L} .. \overline{\mathrm{x}_1} \in \mathcal{A}^L$ where $\{\bar{\mathtt{A}} = \mathtt{T}, \bar{\mathtt{C}} = \mathtt{G}, \bar{\mathtt{G}} = \mathtt{C}, \bar{\mathtt{T}} = \mathtt{A}\}$

The complete set of DNA of an organism is called its *genome*. Genome sizes vary widely according to the organism. A bacterial genome is typically composed of one long circular chain, 100Kbp - 10Mbp long. For the human genome this number stands at about 3.2 billion bp. As for most eukaryotes, the genome is spread onto several linear chromosomes.

Function, and link with RNA and proteins

Some regions of the genome, called *genes*, are the blueprint to manufacture *proteins*, the main functional macromolecules in organisms. They are themselves chains of elementary molecules, amino-acids, which are directly mapped to the gene they come from. Not all DNA is part of

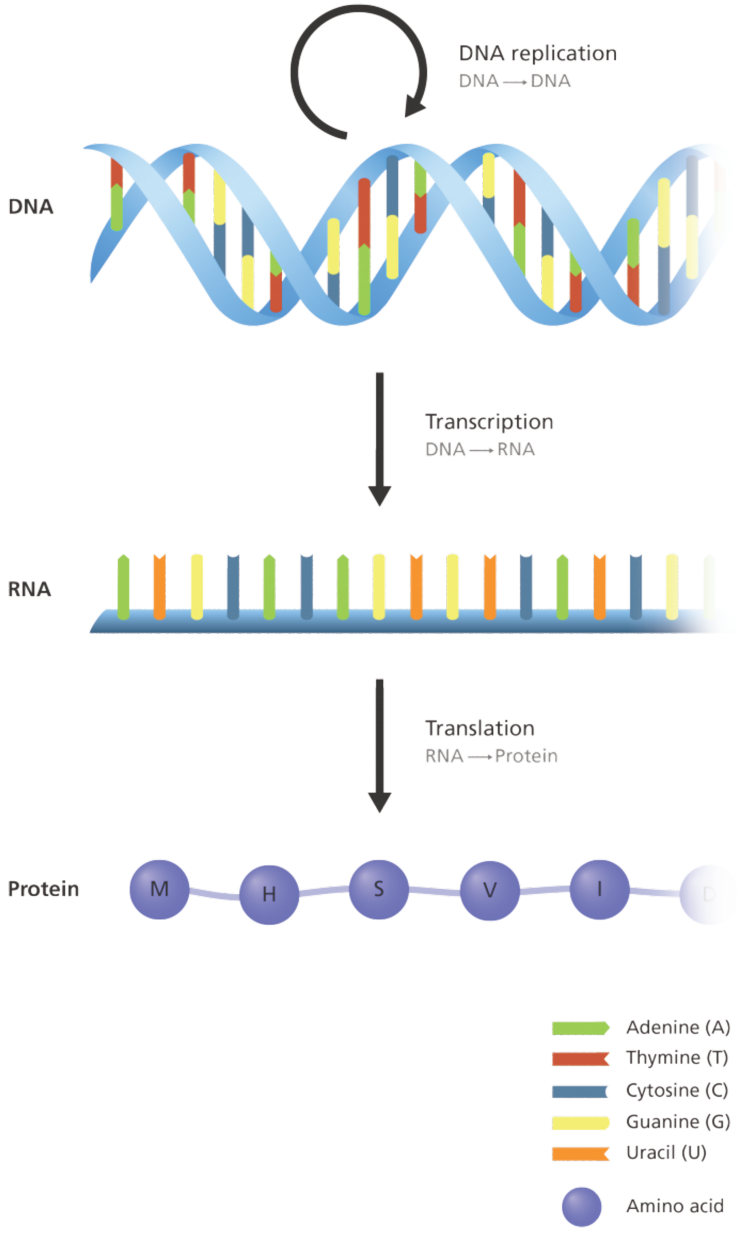


Figure 1.1 – The "Central Dogma" of molecular biology. Image credit: Genome Research Limited

gene: some parts act as signals for other chemical processes (transcription factor binding), others are obsolete leftovers of ancient genes , and others we still don't fully understand. In prokaryotes, genes constitute most of the genome sequence (between 85% and 90% [Koonin, 2011]), with the rest. In humans, however, only 1% of the genome is coding DNA.

As an intermediary step to the synthesis of proteins from genes, the DNA region of interest is copied onto another macromolecule, RNA, in a process known as *transcription*. Very similar to DNA, it is a single-stranded chain of the nucleic acids `A`, `C`, `G` and Uracil `U` (replacing `T`). It is then the messenger RNA that are transported to the cell's factory, the ribosome, and *translated* into proteins (Figure 1.1). Not all genes are constantly transcribed into RNA (expressed). Gene expression varies according to the type of cell, its environment and its age. For instance in humans, the gene coding for hemoglobin is only expressed in blood cells. The study of gene expression, and in particular the process of counting and identifying RNA in cells, is called *transcriptomics*.

Evolution

Although stable, the genome of an organism can change through time, either by chance *mutations* during replication, by combination with another organism (sexual reproduction for eukaryotes, or recombination for prokaryotes) or by exposure to an environmental stress. If those mutations are viable they are passed on to offspring, and once a sufficient number of them accumulate, it can give rise to a new species (*speciation*). Albeit these differences, genome sequences within the same species tend to be very similar. For instance if one were to pick two humans from anywhere in the world, their genomes would be on average more than 99% identical [The 1000 Genomes Project Consortium et al., 2015].

DNA *sequencing* is the process of determining (or reading) the DNA sequences from a biological sample. We have known about DNA for more than a hundred years[1], but we have known our own genome only for the past 20.

1.1.4 Whole genome sequencing

The first complete sequencing of a human genome ended in 2001 after 10 years and a billion dollars worth of materials and labor. Barely a decade later, the same work could be done in under a day for about $1000. This drastic improvement in cost and in efficacy was enabled by a technological revolution in DNA sequencing processes and machines, allowing DNA sequencing to be a viable alternative for many applications – metagenomics being one of them [Eisen, 2007]. We will now briefly walk through the state of the art of DNA sequencing, so-called Next Generation Sequencing (NGS; [Goodwin et al., 2016]). Although long-read technologies [Amarasinghe et al., 2020] have a growing number of applications and seem promised to a bright future, the most used platforms for metagenomics and for most other applications as of 2020 are *short read*, or shotgun, sequencers. They do not output the complete DNA sequences of chromosomes, but rather process millions of small DNA fragments in parallel. The NGS pipeline goes roughly as

[1] We are referring here to Thomas Morgan's 1915 work proving that genetic information is carried on the chromosomes. Many other dates could have been selected, from the 1869 discovery and isolation of the DNA molecule by Swiss physician Friedrich Miescher to 1953, when Watson and Crick famously determined its structure and function. We chose a hundred years for dramatic effect.

follows, with field biologists performing step (i) and platform technicians performing steps (ii) through (iv):

(i) **Sampling** from the habitat, filtering particles and extracting the DNA.

(ii) **Shearing** DNA strands are cut into fragments of roughly 50-400 bp. Fragment lengths are given a certain distribution determined by the platform engineer.

(iii) **End ligation**: Special DNA sequences are added to the ends of each fragment. These will be used to put the fragments into position in the machine.

(iv) **Amplification**: Fragments are cloned multiple times and batched into clusters.

(v) **Sequencing**: The short fragments are given to the machine and processed in parallel. By play of luminous and/or chemical signals, bases are read on each cluster. This process takes about 10 hours for a human genome on an Illumina HiSeq, the current leading platform on the market.

(vi) The results are printed out: an output file of billions of short DNA reads in random order.

While NGS technologies were initially designed to sequence an individual organism's genome, they have since been used to sequence whole ecosystems. Indeed, the DNA reads will come from all the different organisms in the sample. The collection of these reads is called the *metagenome*.

Processing the output

In order to make sense of the sequencing reads, which we can think of as millions of puzzle pieces coming from many different puzzles, bioinformatics approaches are needed. Several downstream tasks are possible, depending on the application. The pieces can be reassembled together in a process called genome *assembly*. If no reference genome (i.e. the original puzzle picture) is available this is called *de novo* genome assembly, which is a necessary step to discover genomes for new species. The assembled genome can be compared to reference genomes to identify mutations in process known as *variant calling*.

Alternatively, if we are interested only in the community composition of organisms and not in their full genomes, the pieces can be sorted into buckets corresponding to their genomes of origin (*genome binning*, which does not require the species to be known a priori) or to their taxonomic clade (*taxonomic binning*). Once this step is performed we can detect species of interest, or characterize the community proportions (abundance profiling). A schematic representation of the whole pipeline is shown in figure 1.2.

Sequencing noise

Sequencing machines can make mistakes: they sometimes change a base for another (*substitution*), forget a base or a series of bases (*deletion*) or add nonexistent bases or series of bases (*insertion*). Each machine has its own error-profile and biases. The popular Illumina HiSeq platforms for example have an overall 0.1% chance of substitution [Goodwin et al., 2016]. These probabilities are also not the same for bases according to their position within the read: bases near the end of the read are more often wrong than those at the beginning.

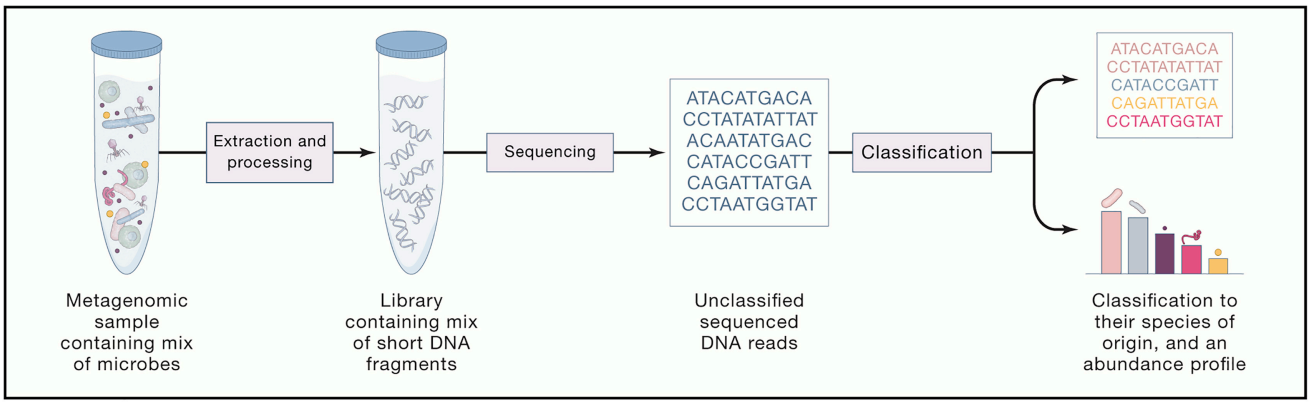


Figure 1.2 – Metagenomics pipeline for microbial community profiling. Figure taken from [Ye et al., 2019]

Several software ([Angly et al., 2012], [Shcherbina, 2014] see [Escalona et al., 2017] for a review) have been developed that replicate these error profiles, and can simulate output from the different sequencing machines. These simulators can be used to generate training or validation datasets [Fritz et al., 2018] to assess bioinformatics software performance.

A major challenge is to differentiate biological mutations and variants from sequencing errors. To help with this issue, some sequencing platforms also return an accompanying quality or confidence score for each base pair.

Other common types of sequencing

Metagenomics was first performed by targeted sequencing of a discriminating gene: the 16S ribosomal RNA (rRNA) gene [Lane et al., 1985]. Present among all bacteria, it has the convenience of having both highly conserved *primers* (short base-pair substrings) that serve as targets so the sequencer can find the gene, and highly variable regions which are species-characteristic. While relatively cheaper and yielding good enough accuracy [Jovel et al., 2016], 16S sequencing was superseded by whole genome sequencing (WGS) for applications requiring more complete information such as functional predictions or a more precise taxonomy. Moreover 16S sequencing can fail to detect organisms from novel lineages that have modified primers [Hug et al., 2016]. Growing evidence of horizontal gene transfer even for the 16S rRNA gene [Miyazaki and Tomariguchi, 2019] also suggest caution when using it to assign species. Also worthy of note, WGS metagenomics allows the study of organisms lacking the 16S gene such as viruses, archaea or even eukaryotes.

For a more complete characterisation of microbial communities, DNA sequencing can also be complemented by other 'omics' techniques such as transcriptomics (which genes are expressed) RNA-Seq, or proteomics (which proteins are produced), ... These techniques can be performed with the same machines.

Now that we have a complete picture of the field of metagenomics, we will specifically focus on methods for the taxonomic binning of short shotgun reads. Although not specifically suited to them, these methods can also be used for the other tasks of abundance profiling and organism detection.

1.2 Taxonomic Binning

1.2.1 Presentation

Setting

Although the biological and technological background may be complex, in the end the binning problem is fairly simple to formulate: we must assign a label to small fragments of DNA. This can be thought of as a DNA Shazam: given a small sound sample one must find which "song" it is from.

From this point onward we will assume the T possible outcome classes – bacterial species for example – are labelled from 1 to T. A taxonomic *binner* is a function f that maps a DNA read $\mathbf{x} \in \mathcal{A}^L$ to a taxonomic label $i \in \{1, ..., T\}$.

In order to calibrate the binner, we assume that we have a set of annotated reference genomes $\{(G_i, y_i),\ y_i \in \{1, ..., T\}\}$. As a point of reference, as of 2020 there are 13000 such genomes publicly available on the website of the National Center for Biotechnology Information (NCBI) [NCBI Resource Coordinators, 2016]

Methods overview

With few exceptions, all methods for taxonomic binning discussed below rely on matching small patterns or sub-sequences (also called *seeds*) to the reference genomes.

- *Aligners* try to align the read locally around the seed match.
- *Pseudoaligners* use discriminative seeds so that once a seed is found to match to a genome G_i, they directly return G_i (or its label y_i).
- *Compositional* approaches learn representations for each of the seeds then combine these representations to classify the read.

In order to quickly find seed matches, the reference genomes are indexed into a database that comes conceptually in one of these two forms:

(i) A *hash table*, which enables efficient lookup of seeds of fixed size. Any type of information can be stored with the seed. Hash tables are fast but memory-intensive.

(ii) A *full-text index*, either a suffix array or an FM-index. The FM-Index [Ferragina and Manzini, 2000] is a data structure based on the Burrows Wheeler transform (BWT) [Burrows and Wheeler, 1994] that allows both compression of reference genomes and constant time lookup of sub-sequences of arbitrary size.

We will review these methods in more detail in the following sections, but first we give the definitions for typical seeds.

Definition. *A* k***-mer*** $\mathrm{x} \in \mathcal{A}^k$ *is a sequence of* k *nucleotides. The* k*-mer decomposition of a sequence* $\mathbf{x} \in \mathcal{A}^L$ *of length* L *is the collection of all the* $L - k + 1$ k*-mers appearing in* $\mathbf{x}$*.*

Ex: The 3-mer decomposition of `AATGGCA` is $\{$`AAT`, `ATG`, `TGG`, `GGC`, `GCA`$\}$

Definition. *Given a k-mer ordering (lexicographic for instance), and two integers l and k such that $k < l$, the **minimizer** of an l-mer u is the smallest k-mer occurring in u. The (l, k)-minimizers of a sequence $\mathbf{x} \in \mathcal{A}^L$ are the minimizers of all the l-mers in $\mathbf{x}$.*

Ex: The $(5, 3)$-minimizers of `AATGGCA` are $\{$`AAT`, `ATG`, `GCA`$\}$

Definition. *A (l, k)-**gapped** k-mer (or spaced seed) is a non-contiguous subsequence of k characters of an l-mer.*

Ex: `A_TGGC_A` is an (8,6)-gapped k-mer.

Wisely chosen gapped k-mers have been shown to be better than regular k-mers for certain applications [Leslie et al., 2003]; [Ghandi et al., 2014]; [Brinda et al., 2015] as they are more robust to noise and can model longer dependencies.

1.2.2 Alignment-based methods

General principle

Sequence alignment, or *read mapping* is the process of finding the originating position of a read in its parent genome.

BLAST [Altschul et al., 1990] was the go-to algorithm for sequence alignment, until the advent of NGS and short reads brought the need for faster algorithms. Most state-of-the-art aligners work on some variation of the *seed-and-extend* paradigm. Exact short sub-sequences (*seeds*) from the query read are looked up in the reference genome. This lookup step should be quite fast. Once matching seeds have been found, they are extended (*local alignment*) by a dynamic programming algorithm (Needleman-Wunsch, Smith-Waterman), allowing mismatches. The seed size needs to be adapted: if it is too short there will be many matches and therefore many calls to the expensive local alignment step. On the other hand if the seeds are too large there is a risk of not finding any exact matches.

BWA-MEM [Li, 2013] and closely related Bowtie 2 [Langmead et al., 2009a] use the BWT transform and allow for lookups of arbitrary size with a certain number of mismatches. Longer seed matches can be found when they exist, reducing the chances of unsuccessful extensions or of redundant matches. The long read successor to BWA, Minimap2 [Li, 2018] uses a hash-table with seeds of fixed size instead (k is usually between 10 and 20), trading those benefits for more efficient computation.

For a complete review of alignment algorithms see [Canzar and Salzberg, 2017].

Common issues and extensions

Although very precise, these alignment approaches are relatively slow and do not scale well with the growing number of reference genomes. Several approaches have been proposed to answer these difficulties.

A first class of methods reduce the number of reference genomes by first performing a coarse search. After finding candidate reference genomes according to their global similarities with the sample metagenome, they use standard alignment tools to align the reads to these candidates. For instance [LaPierre et al., 2020] compare the global k-mer content of the metagenome to find candidates. This two-step procedure can be done with any other algorithm as the first coarse search.

A second class of methods such as taxMaps [Corvelo et al., 2017] compress the reference database to remove parts that are redundant for taxonomic classification. [Subrata et al., 2020] first select discriminating unique regions of reference genomes and use these as reference. Other software such as MetaPhlan2 [Truong et al., 2015] and mOTUs2 [Milanese et al., 2019] go even further and only retain certain marker genes. These methods will not find matches for a great portion of reads, so cannot be used for taxonomic binning per se but can be used for abundance profiling or species detection.

Although alignment methods can be used for taxonomic binning, they are not specifically suited to the problem, as we do not need to know the originating position of the read in the reference, but only if it does occur. Methods in the following section bypass the local alignment (*extend* part) and only focus on the seeding.

1.2.3 Pseudoalignment

Pseudoalignment consists in memorizing long discriminative subsequences (k-mers with $k > 20$), along with their taxonomic label. Kraken [Wood and Salzberg, 2014] stores all the k-mers it has seen in the training set in a big hash table. Each k-mer entry stores the taxonomic label y_i of the genome it comes from. If it is seen in several genomes G_i and G_j, the label of the least common ancestor (LCA) $y_i \cap y_j$ of G_i and G_j is stored. To classify a read, Kraken tries to find each of the k-mers in the reference table, then returns the label that has the most matches. Kraken 2 [Wood et al., 2019] improves the data structure into a Compact Hash Table with lower memory usage, and uses $(35, 31)$-minimizers as seeds. CLARK is a very similar algorithm that only stores unique k-mers (those that appear in only one clade), reducing database size.

[Schaeffer et al., 2015] have a sensibly similar method (applied to the analogous problem of RNA quantification), being more refined by matching k-mers to reference *contigs* (see section 1.5.2) instead of taxonomic clades, and using an EM procedure to estimate the abundances. As a taxonomic binner it is more or less equivalent to Kraken, but the contig matching enables finer-grained abundance estimation.

Centrifuge [Kim et al., 2016] is a hybrid approach that stores a highly compacted BWT index and matches variable-size k-mers to this index. k-mer hits with maximal k (Maximal Exact Matches: MEM) $k \geq 22$ are recorded then scored using an empirical formula (score species $= \sum_{hits} (\text{hit length} - 15)^2$) to find the best matching species.

All of the above methods are orders of magnitude faster than pure alignment strategies, while being of comparable accuracy. A disadvantage of their reliance on long seeds though is that reads with no seed matches are left unclassified. If the input data is noisy or from species remote from the reference database, a significant proportion of reads can end up unclassified.

1.2.4 DNA-to-protein

Some software such as Kaiju [Menzel et al., 2016] or MMseqs2 [Mirdita et al., 2020] first translate the DNA reads into proteins then match them to protein reference databases. This is a viable approach for two reasons. Firstly this has the theoretical advantage of being more robust to biological noise. Because of the degeneracy (redundancy) of the amino-acid mapping, protein sequences are more conserved than their underlying DNA. For example both the triplets `AAA` and `AAG` code for the same amino-acid Lysine, so a `G` $\leftrightarrow$ `A` substitution error on the third nucleotide should have no observable effect, and will not be weeded out by evolution (this is called a

synonymous mutation). Second, a large portion (>80%) of bacterial genomes are translated into proteins. Kaiju functions essentially like Centrifuge in protein-space, returning the genome matching the longest MEM (minimum length m=11 aminoacids) instead of weighted average over all MEMs.

1.2.5 Binner Evaluation

Three criteria are used for evaluation:

(i) The *classification accuracy* is the proportion of reads that are correctly classified. There are different measures of accuracy: averaging the accuracy over the whole sample will be different than if averaged over each class. These metrics must be balanced according to the application. A small number of false positives (classifying reads as species that were not present in the sample) for example is not too problematic for abundance profiling, but can be very misleading for pathogen detection.

(ii) The classification *speed* is the number of reads that can be classified per second. For large binners, the fixed cost of loading the binner into memory can be significant, although amortized by the number of reads classified. Other fixed time-costs include the time it takes to build the binner database from the reference genomes. For machine learning algorithms this is the model training time. Since this step is performed only once there is not much need for it to be fast.

(iii) The amount of *memory* a binner takes up, both in disk space and in RAM. The RAM in particular can be a limiting factor, both for building databases and at prediction time, making some algorithms impractical for regular computer hardware.

For a review of the performances of these methods, see [Sczyrba et al, 2018] or [Ye et al., 2019]. As a rule of thumb, alignment is precise but slow, pseudoalignment is very fast but come at a high memory cost. Methods from both classes can return a lot of unclassified reads, hurting some accuracy metrics.

One disadvantage with these approaches is that they do not make use of probabilities and of training set frequencies. Several seed matches do not interplay in a sophisticated way. Compositional approaches and statistical learning ones in particular are natural tools to make these features interact in a more robust manner.

1.3 Supervised Learning Framework

The taxonomic binning problem or more generally any systematic label assignment to a biological sequence can be framed as a supervised learning problem.

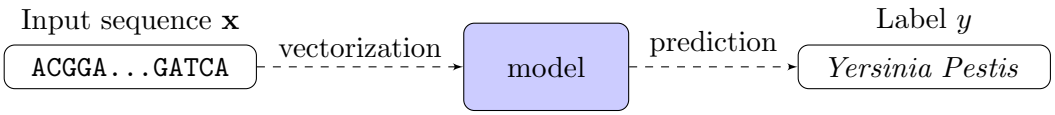


Figure 1.3 – Schema for taxonomic binning, and biological sequence classification in general

Goal

The field of supervised learning aims to predict an unknown characteristic of a sample, the *label*, from a set of observable features of this sample. This could be a wide variety of problems, such as identifying the speaker in an audio sample, predicting tomorrow's weather or labelling a patient's susceptibility to disease from his genetic information.

Let $\mathcal{X}$ and $\mathcal{Y}$ be the input (feature) and output (label) spaces, respectively. It is assumed that there is an unknown joint distribution $\mathcal{P}$ over $\mathcal{X} \times \mathcal{Y}$. The goal of supervised learning is to approximate the marginal distribution $\mathcal{P}(\mathcal{Y}|\mathcal{X})$: given an input $x \in \mathcal{X}$, what should the output y be. The distribution $\mathcal{P}$ is only indirectly observed through a series of sample points (x_i, y_i) drawn from $\mathcal{P}(\mathcal{X}, \mathcal{Y})$: the *training data*.

The aim is then to teach a predictor $\hat{f} : \mathcal{X} \to \mathcal{Y}$ such that $\hat{f}(\cdot)$ approximates $\mathcal{P}(\mathcal{Y}|\cdot)$. In the cases where the output is categorical ($\mathcal{Y}$ is a discrete set), predictors can output a probability distribution over the possible outcomes y.

General method overview

To find this predictor $\hat{f}$, we first need to define a functional space $\mathcal{F}$ in which to search for. In order to facilitate searching, $\mathcal{F}$ is often parametrized by a set of parameters $\theta \in \mathbb{R}^p$: $\mathcal{F} = \{f_\theta, \theta \in \mathbb{R}^p\}$. When $\mathcal{X}$ is a vector space of dimension N, a common very simple choice for $\mathcal{F}$ is the set of linear functions $\mathcal{F}_{linear} = \{x \to \theta \cdot x, \theta \in \mathbb{R}^N\}$. The "size" of the span of all the f_θ is called the model *capacity* or *complexity*.

Once this space is defined, we must find the "optimal" $\hat{f}$. Optimality in this case is defined with respect to a *loss function* $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ that penalizes bad predictions and rewards good ones: $\ell(\hat{y}, y)$ is large if the estimation $\hat{y}$ is far from the true value y and low otherwise. The optimal predictor f with respect to loss ℓ is the one that minimizes the theoretical *generalization error*:

$$\mathcal{L}(f) = \mathbb{E}_{\mathcal{P}}\left[\ell(f(x), y)\right]. \tag{1.1}$$

Since we do not have access to $\mathcal{P}$, $\mathcal{L}$ cannot be explicitly computed and is approximated by a proxy, the training error or *empirical loss* $\mathcal{L}_{emp}$:

$$\mathcal{L}_{emp}(f) = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \ell\left(f(x_i), y_i\right). \tag{1.2}$$

$\mathcal{L}_{emp}$ is simply the average loss across all the training samples. Supposing we have parametrized the functional search space, finding the optimal predictor then becomes the minimization problem:

$$\min_{\theta} \sum_{i=1}^{N_{train}} \ell\left(f_\theta(x_i), y_i\right). \tag{1.3}$$

In the classification problem with T classes, estimated probabilities $\hat{p} \in \mathbb{R}^T$ can be scored with the the *cross-entropy* loss ℓ_{ce}, a standard loss function we will use later on. If $t \in \{1, \ldots, T\}$ is the true label and $\hat{p}_t$ the estimated probability of that label, the score is

$$\ell_{ce}(\hat{p}, t) := -\log(\hat{p}_t). \tag{1.4}$$

Optimization

Training, or calibrating a model is the procedure of solving (1.3) for the optimal set of parameters θ^*. When derivatives w.r.t. θ exist for the function f_θ, a natural method is to set the gradient $\nabla_\theta \mathcal{L}_{emp}(f_\theta)$ to 0:

$$\nabla_{\theta^*} \mathcal{L}_{emp}(f_{\theta^*}) = \sum_{i=1}^{N_{train}} \partial_1 \ell \left(f_{\theta^*}(x_i), y_i \right) \nabla_{\theta^*} f_{\theta^*}(x_i) = 0. \tag{1.5}$$

In cases where this equation has no closed form solution or is hard to compute (true for most models f_θ), the preferred optimization method is gradient descent. θ is iteratively steered in the direction of steepest descent:

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta \nabla_{\theta^{(i)}} \mathcal{L}_{emp}(f_{\theta^{(i)}}). \tag{1.6}$$

The learning rate η is a carefully chosen optimization hyperparameter that controls the size of the updates. For a large number of training samples N_{train}, computing the gradient at every point x_i can be quite expensive. Stochastic gradient descent (SGD) approximates $\mathcal{L}_{emp}$ by using only a random subset of the data samples (a *batch*) at each iteration.

Representation learning and Neural Networks

A word on designing the functional space $\mathcal{F}$. An almost systematic first step is to (implicitly or explicitly) map the input space to a vector space of dimension N, with a mapping $\phi : \mathcal{X} \to \mathbb{R}^N$. Linear models are a solid starting point for classification – logistic regression; support vector machines (SVMs) – and regression – ordinary least squares. They are easy to optimize but can lack in capacity if the feature space ϕ is poorly designed. *Representation learning*, is the task of choosing the vector representation ϕ so that the downstream tasks can be easily solved by linear methods. Such representations for DNA reads are illustrated in figure 1.4, where the classes are clearly separated in the embedding space. One way to build a complex function is to combine simple functions together. In neural networks (NNs), the mapping function ϕ is the composition of several elementary functions or *layers* $\phi_{NN} = \phi^{(n)} \circ \phi^{(n-1)} \circ \ldots \circ \phi^{(1)}$. Each layer has a simple form $\phi^{(j)}(x) = \sigma\left(\theta^{(j)} \cdot x\right)$, where σ is a nonlinear function and $\theta^{(j)}$ a set of *weights*. These types of functions ϕ_{NN} have the double advantage of being able to approximate arbitrarily complex functions, and to have easily computable gradients thanks to the chain rule. A deep neural network is simply a neural network with many layers n.

As we will see in later sections, types of neural networks differ mostly in how they constrain the weights $\theta^{(j)}$. For instance in recurrent networks the weights are the same for every layer $(\theta^{(1)} = \ldots = \theta^{(n)})$, while convolutional networks on the other hand impose structured sparsity on the matrices θ.

Regularization

Given a limited amount of training data, a high-capacity model can *overfit*: match exactly the training data points $\hat{f}(x_i) = y_i$ while roaming free on all other points of $\mathcal{X}$. Overfitting models have an excellent empirical risk but fall off on the generalization error. To mitigate this a possible approach is to increase the number of training data points. If there are none available, it is

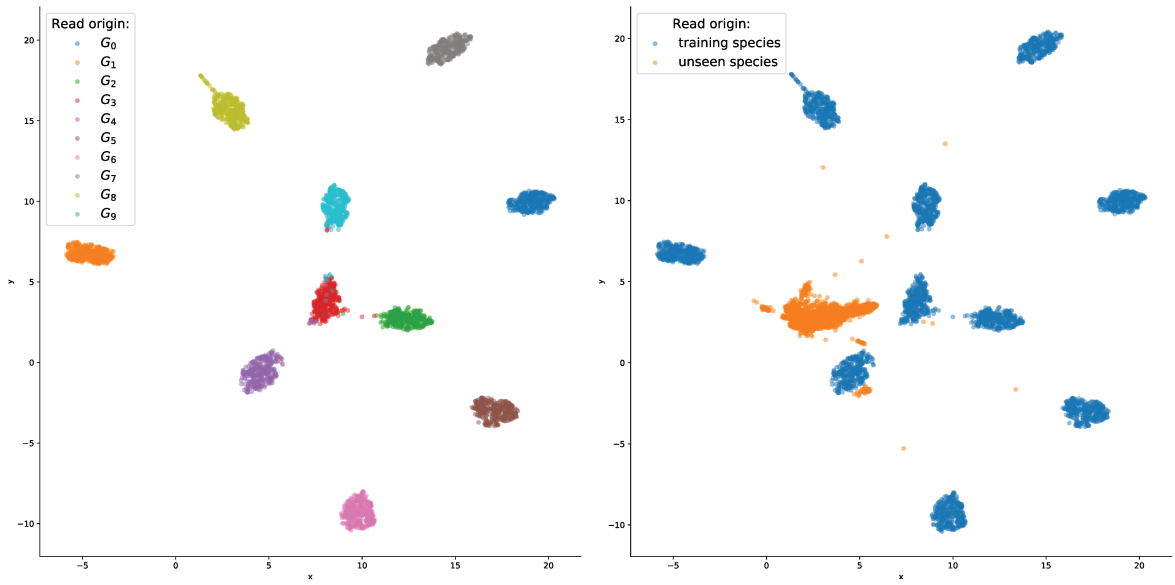


Figure 1.4 – `fastDNA` embeddings for simulated reads. The model was trained on 10 genomes $\{G_i, i = 0 \dots 9\}$ with k-mer size $k = 12$ and an embedding dimension of $d = 10$. The read embeddings were passed to UMAP [McInnes et al., 2018] to reduce the dimension to 2 for plotting. Each point represents a DNA read. On the left, reads from the training genomes, colored by their genome of origin. On the right we add reads from unknown species (in orange) to the same UMAP mapping. The training data is the A1 dataset from [Luo et al., 2017] and the unknown species are from the B1 dataset of the same reference.

sometimes possible to create new ones, hopefully staying true to the distribution $\mathcal{P}$. For example we could apply small deformations to some points x_i while conserving their labels. For instance in image processing, one can shift, rotate or change the light/color of the image and for DNA sequences we can add some small mutations. This process, known as *data augmentation*, often requires expert knowledge about the kind of data involved.

Another way of improving the model's generalization error is to impose smoothness on $\hat{f}$, intuitively justified by the assumption that small changes in input should lead to small changes in output. This can be done explicitly by penalizing high variations with a regularization term $\Omega(\hat{f})$, added to the objective function $\mathcal{L}_{regularized} = \mathcal{L}_{emp} + \Omega$. For example L2-regularization penalizes the magnitude of parameters θ: $\Omega(\hat{f}_\theta) = \lambda \| \theta \|_2$, where $\lambda > 0$ controls the regularization strength.

One can also implicitly regularize by restricting the functional space $\mathcal{F}$. For instance the space of multidimensional linear functions $\mathcal{F}_{linear} = \left\{x \to \theta \cdot x,\, \theta \in \mathbb{R}^{N \times M}\right\}$, although already naturally limited, can be regularized even more by adding a rank constraint on the weights matrix $\text{Rank}\,(\theta) \leq n < \min(N, M)$.

Finally a more recent method of regularizing is to perturb the training procedure by injecting noise in the model parameters [Srivastava et al., 2014]; [Ji et al., 2016].

1.4 Supervised learning for biological sequences

In our taxonomic binning problem, the input space $\mathcal{X}$ is $\mathcal{A}^*$, the set of sequences composed of characters from alphabet $\mathcal{A}$. Again assuming we have labelled the T output classes (species or any other taxonomic unit) from 1 to T, the output space is the categorical $\mathcal{Y} = \{1, ..., T\}$.

We create the training data by extracting short reads from a choice of reference genomes. These genomes are then withheld from the training set. Ideally the simulated community composition should be as close as possible to the unknown communities we are trying to characterize. Sequencing noise and realistic biological noise can be added to the fragments as well.

1.4.1 Range of applications

We give below a set of related problems in biological sequence modelling, where only the alphabet $\mathcal{A}$ changes.

In *proteomics*, the alphabet $\mathcal{A}_{prot}$ is the set of 20 amino-acids composing proteins. Since the function of a protein is mostly determined by its 3d structure, a fertile field of research is predicting protein structure from its sequence. One can also try to directly predict some of its functions, such as its binding to certain molecules or its interactions with other proteins. Understanding the relationship between a protein sequence and its functions has major applications in drug design for instance.

For *transcriptomics* the alphabet is almost the same as the DNA alphabet $\mathcal{A}_{RNA} = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\}$. Given an individual sequence of RNA, we can determine its type, its structure or identify functional capacities such as protein binding or alternative splicing sites. Another domain, very related to taxonomic binning, is the quantification of gene expression. Short RNA reads are mapped to their original gene and then counted.

Finally for *genomics* we have already seen the alphabet is $\mathcal{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Aside from taxonomic binning, there are many tasks centered around predicting a DNA sequence's function, such as recognizing transcription factor binding site (TFBS) or identifying promoter or enhancer regions (regulatory genomics).

Reverse-engineering the relationships between biological sequences and their function is of capital importance for precision medicine and drug design. Indeed we already know how to manipulate, modify and even create these sequences, now careful research must be put into understanding the effect of these manipulations.

1.4.2 Natural Language Processing

Fortunately for us bioinformaticians, biological sequence modelling is akin to another extensively researched domain: natural language processing (NLP) – the study of written human language. Both try to make sense of sequences of characters.

Broadly speaking, all NLP methods first break up the input text S into units or *tokens* $v_i \in \mathcal{V}$, where $\mathcal{V}$ is the chosen vocabulary. This is called word segmentation or tokenization. The tokens can be individual characters, subwords, words or word combinations (e.g. "New York" could be represented as one token.). Each token is given a vectorial representation and is then processed by a model. The model combines those representations and makes a decision based on the combined representation.

There are however some key differences with biological sequences:

(i) The generating processes $\mathcal{P}$ behind both types of data are different. If NLP models were too tailored to human language in their design they could possibly not generalize well to biological sequences.

(ii) Most models are based on the fundamental unit of words, however there is no such concept in DNA, at least not at the sequence lengths we are considering. Other tokenizations: subwords, characters. In order to apply NLP methods to DNA, we must define a notion of word, which is similar to the notion of *seed* used in alignment and pseudoalignment (see previous section).

(iii) Similarly there is no clear notion of punctuation or of the beginning and end in a DNA sequence. Shifting a sequence by 1 or 2 characters should not change its label, so models must be robust to translation.

(iv) The vocabulary for natural language remains tractable, being of the order of a million words for large text corpuses. In metagenomics for large k there can be more than a billion k-mers occurring in the reference genomes.

Nonetheless, the best performing statistical learning methods in genomics have closely followed the main trends in NLP.

1.4.3 Bag-of-words representation

A natural way to encode a piece of text is to represent it as the vector of the word counts it contains. Supposing the vocabulary $\mathcal{V}$ is of size $N = |\mathcal{V}|$, each word w is assigned an index

$i_w \in \{1, ..., N\}$. The *one hot encoding* $\phi_{\text{one hot}}(w)$ of the word w is a N-dimensional vector equal to 1 at coordinate i_w and 0 elsewhere. The bag-of-words representation of a sentence S is then obtained by simply summing (or averaging) the encodings of its words

$$\Phi_{bow}(S) = \sum_{w \in S} \phi_{\text{one hot}}(w). \tag{1.7}$$

Genomics

When dealing with DNA sequences (or RNA/proteins) and choosing k-mers as word units, this embedding is called the k-spectral embedding [Leslie et al., 2002]. Several machine learning models have had success by feeding this representation to a standard classifier such as Naive Bayes [Wang et al., 2007], [Gregor et al., 2016] or a Support Vector Machine (SVM) [Vervier et al., 2016] to bin short reads. [Vervier et al., 2016] in particular reach classification accuracies close to the aligners while being an order of magnitude faster. Other choices of word units are valid as well and can lead to increased performances [Leslie et al., 2003]. For instance [Luo et al., 2017] use the exact same model as [Vervier et al., 2016], with specially designed gapped k-mers as word units. SVMs based on gapped k-mer features are also used in state of the art functional genomics, such as regulatory sequence prediction [Ghandi et al., 2014]; [Blakely et al., 2020].

Treatment of reverse-complement

Since a DNA sequence and its reverse-complement are equivalent for all purposes, machine learning models must treat them equally. There are two ways to do this:

(i) Make the model intrinsically invariant (*RC-symmetric*) to the reverse complement operation. For example in most k-mer-based approaches, a k-mer and its reverse-complement are treated as one: a canonical k-mer. When the subsequent model does not depend on k-mer order, both sequences will have the same representation.

(ii) At training time give both (regular and RC) sequences with the same label. At test time average the probabilities for both sequences: $\hat{f}_{final}(\mathrm{x}) = \frac{\hat{f}(\mathrm{x}) + \hat{f}(\bar{\mathrm{x}})}{2}$. . Both of these predictions are naturally RC-symmetric.

A recent paper [Zhou et al., 2020] suggest that the latter method yields slightly better performing models. However for k-mer-based methods enforcing RC-symmetry brings an added benefit in memory as a representation needs to be learned for only half of the k-mers.

1.4.4 Word embeddings

In a breakthrough paper, [Mikolov et al., 2013a] introduced a new way to assign continuous vectors (embeddings) to words. These embeddings are learned in a self-supervised fashion: the input and output spaces are both set to be equal to the vocabulary $\mathcal{Y} = \mathcal{X} = \mathcal{V}$. The skip-gram model takes a word as input to predict the neighboring words (the context), while the CBOW model does the inverse. The predictor is a fully connected neural network with one hidden layer of dimension d. Once fit on the training data, this hidden layer yields a d-dimensional embedding: $\phi(w) = \phi^1 \circ \phi^0(w)$. This method was inspired by the so-called distributional hypothesis in

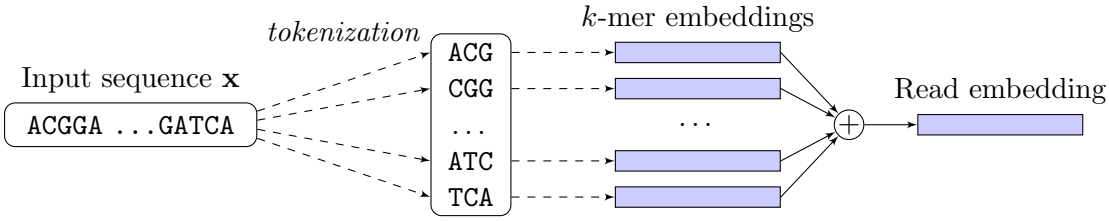


Figure 1.5 – Embedding a read with `fastDNA`. The read embedding is fed to a softmax layer for classification.

natural language which posits that the meaning of words can be deduced from the context in which they appear. It has been shown that these embeddings induce a meaningful geometry on the vocabulary (e.g. $\phi(\text{“}better\text{”}) - \phi(\text{“}good\text{”}) + \phi(\text{“}bad\text{”}) \approx \phi(\text{“}worse\text{”})$). In a followup paper [Bojanowski et al., 2016] introduced `fastText`, which aside from adding subword information gave the option to learn them in a supervised manner. The supervised version of `fastText` proved to be a strong and fast baseline for various language classification tasks [Joulin et al., 2017].

Genomics

Word2Vec-style embeddings were first introduced to genomics and proteomics by [Asgari and Mofrad, 2015], although they did not compare their results to other software.

Our first contribution was to learn these types of embeddings in a supervised manner and to apply them to metagenomics [Menegaux and Vert, 2019]. We showed that they outperformed the best machine learning techniques for taxonomic binning (at the time). Others have since then used a similar model with other k-mer hashing schemes [Shi and Chen, 2019] [Georgiou et al., 2020].

One limitation of such models is their extreme cost in memory as the weights for the embedding layer take up $\mathcal{O}(d|\mathcal{V}|)$ space, where $|\mathcal{V}|$ is proportional to 4^k for regular k-mers. To learn meaningful (competitive) sequence representations for taxonomic binning, k-mer size must be at least 12, which means $|\mathcal{V}|$ quickly scales to the tens or hundred million. Another limitation is that they do not take into account positional information. While k-mer methods are good at finding individual motifs, some DNA functions depend on several motifs being separated by some fixed distance.

Both of these limitations are addressed by the more sophisticated *deep learning* methods which will go over in the next paragraphs.

1.4.5 Convolutional Neural Networks

A convolutional layer ϕ_{conv} works by applying a same function f_{filter} (the convolution filter, or kernel) over all the positions in the input sequence $\phi_{conv}(x) = f_{filter} * x$. This filter is simply a weighted average of the k neighboring positions, where k is the filter size. The filter weights are learnable parameters.

A convolutional neural network (CNN) combines convolutional layers and regular fully connected layers to identify continuous patterns or motifs in the data. CNNs decide themselves which patterns are useful for the task at hand. Although one convolutional layer can only learn simple patterns, as more layers are added the recognized patterns become more and more

sophisticated. CNNs first found success in image processing but then transferred onto many other domains such as audio or text processing [Conneau, 2016]. Applied to text, CNNs either operate on characters or on word embeddings. The former have the advantage of being able to classify substrings that were not seen in training.

In biological sequences, after the characters are one-hot encoded, a read of length L is transformed into a 2-dimensional "image" of size $|\mathcal{A}| \times L$. Character-CNNs have met with great success for a various number of applications in functional genomics [Alipanahi et al., 2015]; [Zhou and Troyanskaya, 2016]; [Kelley et al., 2016]. Deeper CNNs were also used to predict protein folding [Xu, 2019]; [Jones and Kandathil, 2018].

CNNs for metagenomic classification Char-CNNs have shown success in classifying 16S short reads [Busia et al., 2018]. However this problem is of much lesser scale than whole genome sequencing, as the 16S gene sequence is only around 1550 bp long, compared to an order of 1Mbp for a full bacterial genome. [Rojas-carulla et al., 2018] did use CNNs for shotgun sequencing but their performance was subpar. It was later shown in [Liang et al., 2019] that char-CNNs do not work for our problem of full-genome taxonomic binning. One way to make them work is to apply them to a first layer k-mer embeddings, but this defeats the initial purpose of memory savings and of adaptively learning motifs from the data. In fact the performance reported in [Liang et al., 2019] is worse than a simple averaging of embeddings. I presume this is because the number of patterns that need to be learned to distinguish short reads coming from many genomes are too numerous. The needed size of the CNN would outweigh its benefits either in memory or in classification speed.

1.4.6 Recurrent Networks

Recurrent neural networks (RNNs; [Elman, 1990]) process the input sequentially with a same function ϕ,

$$\phi^{(n)}(\mathbf{x}_{[1:n]}) = \phi\left(\mathbf{x}_n,\, \phi^{(n-1)}(\mathbf{x}_{[1:n-1]})\right)$$

keeping a working memory – the *hidden state* – of the results at each step. Well-designed RNNs are able to deal with long range dependencies [Hochreiter and Urgen Schmidhuber, 1997]. RNNs are standard algorithms in machine translation and text generation notably for their ability to generate sequential output [Jozefowicz et al., 2016]. Context-aware word embeddings can be created from the intermediate layers of RNN models [Peters et al., 2018].

[Quang and Xie, 2016] and [Pan et al., 2018] use RNNs in functional genomics by stacking them on top of sequence representations obtained by CNNs.

RNNs are gradually being contested in NLP by a recent innovation: attention [Vaswani et al., 2017].

1.4.7 Transformers

A presumed limitation of Word2vec-style word embeddings is that learning a unique representation per word fails to account for polysemy. One word may have a completely different meaning in one sentence and the next ("robbing the *bank*", "down by the river *bank*"), but yet will have the same representation in both. The attention mechanism was introduced to alleviate this issue [Bahdanau et al., 2015]. Words w are given three separate embeddings:

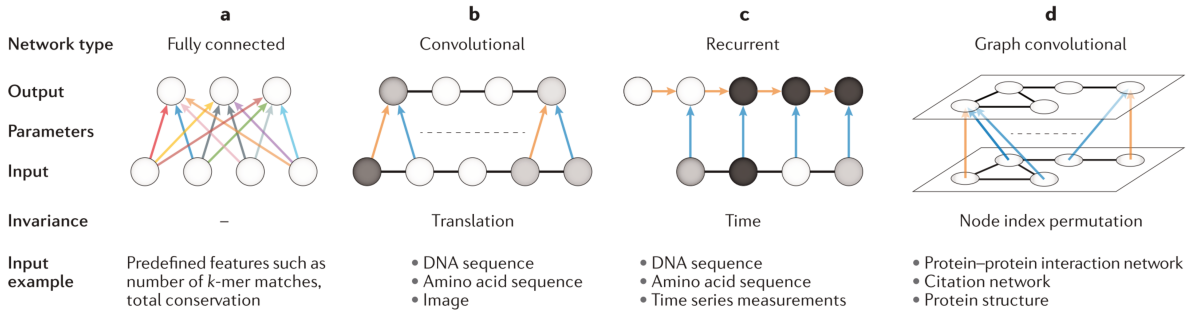


Figure 1.6 – Neural network layers and their parameter-sharing schemes. Neural network architectures can be categorized into four groups based on their connectivity and parameter-sharing schemes. **a** Fully connected layers assume that input features do not have any particular ordering and hence apply different parameters across different input features. **b** Convolutional layers assume that local subsets of input features, such as consecutive bases in DNA, can represent patterns. Therefore, the connectivity and parameter-sharing pattern of convolutional layers reflect locality. **c** Recurrent layers assume that the input features should be processed sequentially and that the sequence element depends on all the previous sequence elements. At each sequence element, the same operation is applied (blue and orange arrows), and the information from the next input sequence element is incorporated into the memory (orange arrows) and carried over. **d** Graph convolutional networks assume that the structure of the input features follows the structure of a known graph. The same set of parameters is used to process all the nodes and thereby imposes invariance to node ordering. By exploiting the properties of the raw data, convolutional neural networks, recurrent neural networks and graph convolutional layers can have drastically reduced numbers of parameters compared with fully connected layers while still being able to represent flexible functions. The same colours indicate shared parameters, and arrows indicate the flow of information. Full lines indicate specific ordering or relationships between features represented as nodes (parts a–d). Figure and caption both from [Eraslan et al., 2019]. License number 4996071460551.

(i) the "value" $v(w)$ – its intrinsic embedding;

(ii) the "query" $q(w)$ – what information it looks for in other words;

(iii) the "key" $k(w)$ – how it is seen by other words.

The *attention score* $\alpha(w_1, w_2) \propto q(w_1) \cdot k(w_2) \in \mathbb{R}$, that captures how relevant w_2 is to w_1. The final embedding of a word w_t is a weighted sum of the values v of the words in its context S (all the words appearing in the sentence for example):

$$\phi^{attention}(w_t, S) = \sum_{w \in S} \alpha(w_t, w)\, v(w)$$

.

The Transformer architecture [Vaswani et al., 2017] uses a special type of attention mechanism, along with a positional embedding for tokens. The embeddings are learned in a self-supervised manner by masking some of the tokens of training sentences and training the model to successively guess the missing tokens (similar to the CBOW model discussed earlier). Most networks using this architecture use a vocabulary of subwords, (such as Wordpiece [Wu et al., 2016]) created adaptively from the training data. State of the art models use Transformer layers and achieve astounding results in most NLP fields such as text translation, generation or classification [Devlin et al., 2017] [Radford et al., 2018].

Transformers have been used with success in proteomics, a closely-related domain, to predict protein structure and form [Rives et al., 2020] [Senior et al., 2020], outperforming other methods by a wide margin. [Ji et al., 2020] introduced DNA-BERT with seemingly improved performances in functional genomics. For taxonomic binning [Liang et al., 2019] use an RNN and an attention mechanism on top of an initial k-mer embedding layer. They show that adding the recurrent layer and the attention mechanism both improve classification performance. To the best of our knowledge, this is the only paper that has shown that going deep gave an improvement over simply averaging word embeddings. However the performance increase is only slight, and it is unclear that the added complexity of the extra layers (restricting k-mer size to 12) outweighs their benefits.

1.4.8 End to end learning

As some DNA regions are shared across different species, some reads may have multiple exact matches. The chance of this happening get slimmer for longer reads but it still can happen. In this case a classifying method that takes into account only the read itself but not its co occurring reads in the sample cannot determine the taxon of origin with certainty. It is only by combining (cross-referencing) information from other reads that . This is the main reason that some approaches take the whole metagenomic sample as input. If 80% of sample is attributed to species A, and the other 20% matches with species A or B indifferently, then with good confidence all of those 20% actually come from species A as well. Several methods have been proposed to disambiguate multiply matched reads a posteriori, including Expectation-Maximization (EM) algorithms [Bray et al., 2016] or bayesian reassignment of reads [Lu et al., 2017]. Multiple Instance Learning [Zaheer et al., 2017] is a field of machine learning specifically suited to this problem, and has been proposed by several recent papers as a robust method making use of probabilities. Recent deep-learning techniques address the whole metagenome as an input, and solve their task directly from end to end [Queyrel et al., 2020], [Georgiou et al., 2020].

1.5 Contributions

This thesis is geared towards embedding short DNA sequences in a low-dimensional continuous vector space, with the end goal of assigning them to taxonomic clades. We first introduce a new model `fastDNA` in which we find embeddings for short k-mers then represent a read as the average embedding of its constituent k-mers. `Brume` expands on `fastDNA` by using the de Bruijn graph to allow larger k-mers. Finally we introduce a custom structured loss for taxonomic classification based on the phylogenetic tree. This loss is applicable to any other deep learning model.

1.5.1 fastDNA

As our first contribution, we proposed a new machine learning model `fastDNA` for the classification of short DNA sequences. The model was trained and applied to taxonomic binning on metagenomics datasets, but could be used for any other biological sequence classification task.

Model

Keeping the notations introduced in the previous section, the model for `fastDNA` is a 3-layer fully connected neural network:

$$\phi_{\texttt{fastDNA}} = \phi^{softmax} \circ \phi^{embedding} \circ \phi^{encoding}$$

The input layer $\phi^{encoding}(\mathbf{x}) : \mathcal{A}^* \to \mathbb{R}^{|\mathcal{V}|}$ is the k-spectral embedding, or one hot encoding of the k-mers in $\mathbf{x}$. The output dimension d of the embedding layer $\phi^{embedding} : \mathbb{R}^{|\mathcal{V}|} \to \mathbb{R}^d$ is a hyperparameter set by the user. The weights $M \in \mathbb{R}^{|\mathcal{V}|} \times \mathbb{R}^d$ of $\phi^{embedding}$, the embeddings matrix, associate a d-dimensional vector to all the k-mers in $\mathcal{V}$. Finally the linear classifier $\phi^{softmax} : \mathbb{R}^d \to \mathbb{R}^T$ is a softmax function over the T output classes. The first two layers are shown schematically in figure 1.5.

The weights of both the linear model and the embedding matrix are learned jointly by stochastic gradient descent with the cross-entropy loss ℓ (1.4).

Evaluation

To train the model we choose reference genomes G_i and their species y_i and simulate on-the-fly reads coming from random positions in these genomes. Random mutations are added to the reads for data augmentation and regularization. This has the effect of bringing embeddings of similar k-mers (similarity here is quantified here as the probability one k-mer can be obtained from the other with mutations) closer together in embedding-space.

Evaluated on datasets from [Vervier et al., 2016], we showed that `fastDNA` outperformed state of the art machine learning approaches and reached equivalent accuracies to alignment software, while being an order of magnitude faster. We also showed that adding some level of chance mutations during training improved the classification accuracy.

Improving memory usage

We show in the work that classification performance continues to increase with larger k-mer sizes. However k is constrained by a memory issue. Indeed the model size scales exponentially with

k as, for the range of k considered, $|\mathcal{V}|$ is proportional to 4^k (on large enough datasets, k-mers with $k<15$ are *dense*, they almost all appear at least once). There are two possible directions to reduce the size in memory of the embeddings matrix.

(i) The first one is to keep the same number of words in the vocabulary but compressing the size of the embeddings. A method of this sort is in fact already implemented in `fastDNA`, which can perform a version of product quantization [Joulin et al., 2016]: a clustering algorithm that preserves dot products between quantized vectors. Quantizing the model in this way reduces the memory size tenfold while having a negligible impact on performance. As a downside, the compression only works post-training, so the full working memory is still needed to calibrate the model.

(ii) The second direction of improvement is to reduce the effective size of $|\mathcal{V}|$, either by pruning out uninformative k-mers (too common or too rare for instance) or by grouping "similar" k-mers and giving them the same embeddings. The similarity could be based on their character content, in which case techniques such as locality-sensitive hashing (LSH) have been explored [Shi and Chen, 2019], [Georgiou et al., 2020]. On the other hand we could define a k-mer similarity or equivalence in a data-driven fashion, and impose that k-mers that always appear together in reads should have the same embedding. For instance one could argue that the k-mers "New Yor" and "ew York" carry equivalent information for classification. This is the focus of our next work.

1.5.2 Brume

The *de Bruijn graph* (dBG) is a data structure that encodes sequences of symbols. It was first introduced to computational biology for genome assembly [Idury and Waterman, 1995] and has since become a staple of the field.

The k-mer based dBG G of a set of sequences $\mathcal{S} = \{S_i\}$ is a directed graph (V, E) where each vertex is a k-mer present in $\mathcal{S}$. Two vertices are linked by an edge if the k-mers they represent appear consecutively in at least one of the sequences S_i.

A *contig*, or unitig, is a maximal non-branching path of the graph. Contigs are illustrated in 1.7. The k-mers of a contig appear in the same sequences and therefore contain the same information for classification. It is then natural to give the same representation to each k-mer of a contig. By doing so we reduce the effective vocabulary from the set of all k-mers $\mathcal{V}$ to the set of all contigs $\mathcal{V}_C$. When k is small, all $(k+1)$-mers appear in the dataset, therefore the dBG is dense: all vertices are linked to each other and each vertex is a contig $|\mathcal{V}_C| \approx |\mathcal{V}|$. However larger k-mers get sparser, and for $k > 20$ empirically $|\mathcal{V}_C| \ll |\mathcal{V}|$.

Adaptation to fastDNA

We build the dBG of the reference genomes using specialized software bifrost or kallisto [Holley, 2020]; [Bray et al., 2016], and keep only the mapping from k-mers to contig indices $\in \{1, \ldots, |\mathcal{V}_C|\}$. As shown in figure 1.8 we learn embeddings for each contig then embed a read by averaging the embeddings of its constituent contigs, weighted by their number of k-mers in the read. The resulting representation is given to a softmax layer for classification.

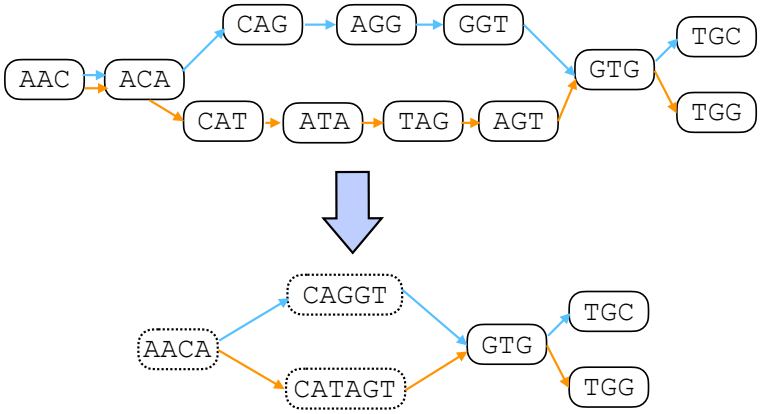


Figure 1.7 – Above: the 3-mer de Bruijn graph of two sequences `AACAGGTGC` (in blue) and `AACATAGTGG` (in orange). Below: Non-branching paths have been merged to yield the compacted de Bruijn graph. Each vertex in this graph is a *contig*. For example the top k-mers `CAG`; `AGG` and `GGT` are mapped to the same contig $C =$ `CAGGT`

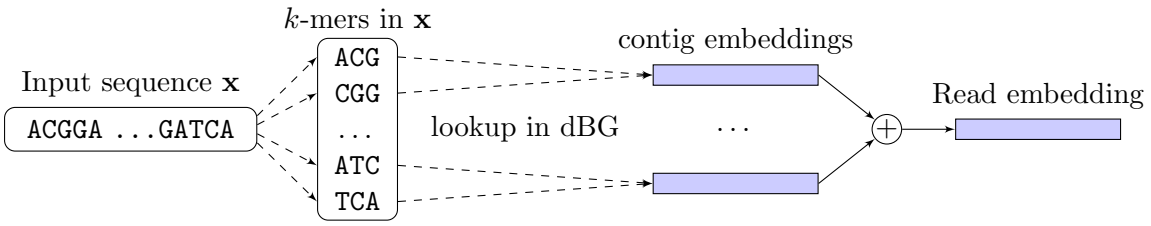


Figure 1.8 – Embedding a read with `Brume`. After tokenization, each k-mer is mapped to its parent contig. The embeddings for each contig are retrieved and averaged. The resulting read embedding is fed to a softmax layer for classification.

Although dBGs have been used for short read classification before [Bray et al., 2016]; [Schaeffer et al., 2015], this is the first attempt to mix them with machine learning. We show that `Brume` is more cautious than `fastDNA`, making fewer mistakes but also classifying fewer reads.

By doing so, we have learned a representation of the compacted de Bruijn graph: each of its vertices has an embedding.

Limitations

Contrary to `fastDNA` with short k, the memory here depends on the total number of k-mers and contigs, which scale with the addition of novel genomes in the database. Current software cannot build de Bruijn graphs for some large datasets without using a prohibitive amount of memory. Furthermore, for large datasets, the number of contigs $|\mathcal{V}_C|$ is still too large. A possible direction of enquiry would be to use larger or more memory-efficient groupings than contigs, such as simplitigs for example [Břinda et al., 2020]. A second concern is that for very large k, k-mers are increasingly rare, most of the times appearing only once in the whole dataset. Hence not only are some reads left unclassified, but this also questions the use of continuous embeddings to represent very rare tokens. We could perhaps consider an adaptive embedding size [Grave et al., 2017]; [Baevski and Auli, 2018].

Another limitation in our current implementation is that the mapping from a read to its constituent contigs is slow. We have not implemented the speedup tricks in [Wood and Salzberg, 2014] or [Bray et al., 2016]. While scaling up to larger datasets we realized that even the vanilla version of `fastDNA` was slower than expected, due to the large number of classes involved. The bottleneck was the softmax layer, and this lead us to explore alternatives.

1.5.3 Phylo-HS, a structured loss for taxonomic classification

The softmax function $\sigma : \mathbb{R}^N \to [0,\, 1]^N$ transforms real scores into probabilities (N is any arbitrary positive integer).

$$\sigma(x) = \frac{e^x}{\sum_{i=1}^{N} e^{x_i}}$$

In neural network terminology, the softmax layer is a linear classifier: the composition of σ with a linear function $z \to \theta \cdot z$. Effectively it gives scores for each class t then normalizes them to sum to 1. In all that follows, we will write z as the d-dimensional input to the softmax layer (the output of the penultimate layer), and $\theta \in \mathbb{R}^{T \times d}$ as its parameters.

$$p_{softmax}(\text{label}\, t \,|\, z) = \frac{e^{\theta_t \cdot z}}{\sum_{j=1}^{T} e^{\theta_j \cdot z}} \tag{1.8}$$

The denominator in (1.8) is also called the partition function $Z(\theta, z)$. Computing this term has a complexity proportional to the number of output classes $\mathcal{O}\,(dT)$, which can quickly become a bottleneck when T gets large, both at test and training time. One way to alleviate this issue is to approximate the denominator by importance sampling [Bengio and Senécal, 2003];[Mikolov et al., 2013b];[Ji et al., 2016]: the sum in Z is taken over a random subset of the samples instead. Although these methods do speed up training, at test time the true softmax must still be evaluated.

Other approaches replace the softmax function by another altogether, such as the hierarchical softmax.

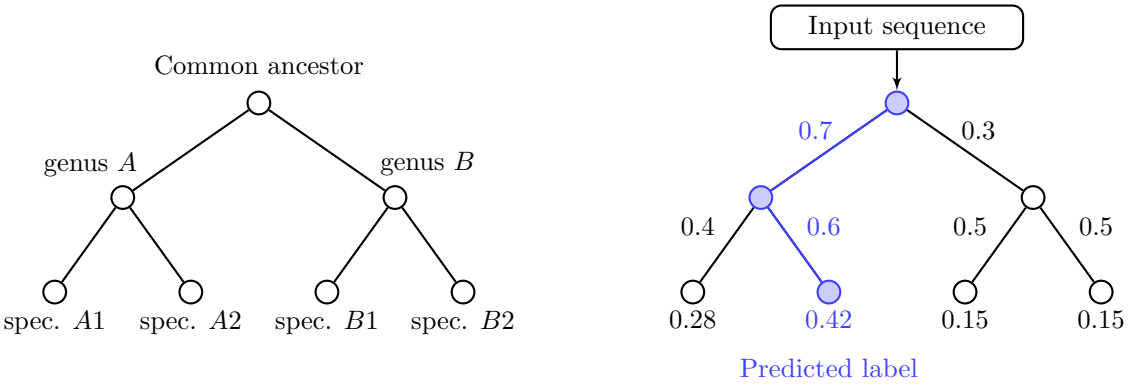


Figure 1.9 – Toy example of a binary classification tree. On the left the labels of interest – species – are ordered in a phylogenetic tree along with their genus. This could be replaced by any other label clustering . On the right, the same tree is used for classification. Each node from the higher levels is an independent binary classification problem, taking the representation $\phi(\mathbf{x})$ of the input sequence. The leaf probabilities are the product of all the edge probabilities. The path of the leaf with highest probability – the predicted label – is highlighted in blue

Hierarchical softmax

The hierarchical softmax (HSM) is an approximation to the softmax function. As initially designed in [Goodman, 2001], the hierarchy was a two-level tree: output labels are grouped into clusters. When given an input $\mathbf{x}$, the model predicts its cluster c with a first model $p(c \,|\, \mathbf{x})$, then predicts the label t with a second model $p(t \,|\, c, \mathbf{x})$. Since each label t belongs to a unique cluster $c(t)$, the probability of t is the product (1.9)

$$p(t \,|\, \mathbf{x}) = p(c(t) \,|\, \mathbf{x})\; p(t \,|\, c(t), \mathbf{x}) \tag{1.9}$$

A simple classification tree is shown on the right of figure 1.9. If the clusters are well balanced, this reduces the complexity of the softmax from $\mathcal{O}\,(dT)$ to $\mathcal{O}(d\sqrt{T})$. Introducing more levels in the hierarchy can further lower the complexity down to $\mathcal{O}\,(d \log T)$ for a balanced binary tree [Morin and Bengio, 2005]. The assignment of labels to clusters can be based either on their similarities or on their frequencies. A speed-optimal HSM based on the Huffman coding of labels was introduced in [Mikolov et al., 2013b]: frequent labels are placed higher up in the tree so that it takes fewer steps to get to them, improving both accuracy and speed.

Although much faster than the standard softmax, the HSM comes at a price in accuracy. One possible explanation is that if similar labels are separated into different clusters, the model might have trouble choosing between the clusters. To correct this, one wants to group similar classes in the same parts of the tree. In the taxonomic binning problem, there is fortunately a very natural way to do this: copy the classification tree on the phylogenetic tree, which, by design, captures similarity between taxonomic classes.

Phylo-HS

Since there are many more taxa in the total phylogenetic tree than in the training set, we take a few steps to prune the tree. We take every label from the training dataset and trace its lineage $l(t)$ to the root of the tree. $l(t)$ is a path $[t_1, ..., t_m]$ where t_1 the root, $t_m = t$; and $\forall i$, t_i is the

parent of t_{i+1}. This collection of paths forms a subtree $\mathcal{T}$ which we further simplify at no loss by merging unique children with their parent. In order to attain maximum speedup we also binarize the resulting tree. This entails making a choice on how to split up siblings into two "artificial" nodes. We choose to split such that the two resulting nodes are as balanced as possible in terms of frequencies (occurrences in the dataset). The resulting tree is thus a hybrid of the phylogenetic tree and the Huffman tree described in [Mikolov et al., 2013b]. It is a *full* binary tree with T leaves and $T-1$ intermediary nodes.

We show that the resulting loss – Phylo-HS – outperforms the Huffman HSM for taxonomic binning in terms of accuracy, while maintaining comparable speed.

1.5.4 Published work appearing in this thesis

The contributions in this thesis are available as published articles or preprints.

- R. Menegaux and J.-P. Vert. Continuous Embeddings of DNA Sequencing Reads and Application to Metagenomics. *J Comput Biol*, 26(6):509–518, 06 2019
- R. Menegaux and J.-P. Vert. Embedding the de bruijn graph, and applications to metagenomics. *bioRxiv*, mar 2020. doi: 10.1101/2020.03.06.980979

The two first chapters closely follow these references. The third chapter contains unpublished preliminary results.

The code for these three chapters is made available on GitHub at `https://github.com/rmenegaux/fastDNA` on the `master`, `kallisto` and `taxonomy_tree` branches respectively. Functions are provided to train and evaluate the models on custom data or to reproduce the presented results.

Chapter 2

fastDNA: Continuous Embeddings for DNA reads

Contents

2.1 Introduction 31
2.2 Method 32
2.2.1 Embedding of DNA reads 32
2.2.2 Learning the embedding 33
2.2.3 Implementation 34
2.2.4 Regularization with noise 34
2.3 Experiments 34
2.3.1 Data 34
2.3.2 Reference methods 35
2.3.3 Small dataset 35
2.3.4 Large dataset 38
2.3.5 Classification speed 39
2.4 Conclusion 40

Abstract

We propose a new model for fast classification of DNA sequences output by next generation sequencing machines. The model, which we call `fastDNA`, embeds DNA sequences in a vector space by learning continuous low-dimensional representations of the k-mers it contains. We show on metagenomics benchmarks that it outperforms state-of-the-art methods in terms of accuracy and scalability.

Résumé

Nous proposons un nouveau modèle pour la classification rapide de séquences d'ADN issues du séquençage prochaine génération. Le modèle, que nous nommons `fastDNA`, représente les séquences d'ADN dans un espace vectoriel en apprenant des représentations continues de basse dimension pour chacun des k-mers qu'elle contient. Nous montrons dans des expériences sur des benchmarks de métagénomique que cette méthode obtient de meilleures performances que l'état de l'art, à la fois en précision et en vitesse de classification.

2.1 Introduction

The cost of DNA sequencing has been divided by 100,000 in the last 10 years. With less than $1,000 to sequence a human-size genome, it is now so cheap that it has quickly become a routine technique to characterize the genome of biological samples with numerous applications in health, food or energy. Besides the genome, many techniques have been developed to measure other molecular informations using DNA sequencing, e.g., gene expression using RNA-seq, protein-DNA interactions using ChiP-seq, or 3D structural informations using Hi-C, to name just a few. In short, DNA sequencing is the swiss army knife of modern genomics and epigenomics. As a consequence, the rate of production of DNA sequences has exploded in recent years, and the storage, processing and analysis of these sequences is increasingly a bottleneck.

While new, so-called *long-read* technologies are under active development and may become dominant in the future, the current market of DNA sequencing technologies is dominated by so-called *next-generation sequencing* (NGS) technologies which break long strands of DNA into short fragments of typically 50 to 400 bases each, and "read" the sequence of bases that compose each fragment. The output of a typical DNA sequencing experiment is therefore a set of millions or billions of short sequences, called *reads*, of lengths 50∼400 in the $\{A, C, G, T\}$ alphabet; these billions of reads are then automatically processed and analyzed by computers to get some biological information such as the presence of particular bacterial species in a sample, or of a specific mutation in a cancer.

Standard pipelines to process the raw reads depend on the target applications, but typically involve discrete operations such as aligning them to some reference genome using string algorithms. In this paper, we investigate the feasibility of directly representing DNA reads as continuous vectors instead, and replacing some discrete operations by continuous calculus in this embedding.

To illustrate this idea, we focus on an important application in metagenomics, where one sequences the DNA present in an environmental sample to characterize the microbes it contains Group et al. [2009]; Riesenfeld et al. [2004]. An important problem in metagenomics is *taxonomic binning*, where each of the billions of sequenced reads must be assigned to a species, given a database of genomes characteristic of each species considered Mande et al. [2012]. Standard computational approaches for taxonomic binning try to align each read to a reference sequence database with sequence alignment tools like BLAST [Huson et al., 2007] or short read mapping tools such as BWA [Li and Durbin, 2009] or BOWTIE [Langmead et al., 2009b]. However, the computational cost of these techniques becomes prohibitive with current large sequence datasets. Alternatively, compositional approaches rephrase the problem as a multiclass classification problem, and employ machine learning methods such as a naive Bayes (NB) classifier [Parks et al., 2011; Wang et al., 2007] or a support vector machine (SVM) [McHardy et al., 2007; Patil et al., 2012; Vervier et al., 2016] after representing each read by the vector of k-mer[1] counts it contains. Interestingly, Vervier et al. [2016] showed that compositional approaches can be competitive in accuracy with alignment-based methods, while maintaining a computational advantage, by using large-scale machine learning approaches. However, a good accuracy is only achieved with k-mers of length at least $k = 12$, corresponding to representing each read as a sparse vector in $N = 4^k$ dimensions. This representation raises computational challenges both at training time (Vervier et al. [2016] push VowPal Wabbit to its limit) and at test time (a $N \times T$ matrix of weights must be stored to model T species).

[1] A k-mer is a contiguous subsequence of k letters.

In this work, we propose to extend state-of-the-art compositional approaches by embedding the set of DNA reads to $\mathbb{R}^d$, with $d \ll N$. For that purpose, we still extract the k-mer composition of each read, but replace the N-dimensional one-hot encoding of each k-mer by a d-dimensional encoding, optimized to solve the task. This approach is similar to, e.g., the `fastText` model for natural language sequences of Bojanowski et al. [2017]; Joulin et al. [2016] or `word2vec` Mikolov et al. [2013a], with a different notion of *words* to embed, and a direct optimization of the classification error to learn the representation. This can reduce the memory requirements to store the model and accelerate classification time when $d < T$, since the $N \times T$ matrix of weights is replaced by a $N \times d$ matrix of embeddings, and a $d \times T$ matrix of weights.

After presenting in more detail the model and its optimization, we experimentally study the speed/performance trade-off on metagenomics experiments by varying the embedding dimension, and demonstrate the potential of the approach which outperforms state-of-the-art compositional approaches.

2.2 Method

2.2.1 Embedding of DNA reads

Given the alphabet of nucleotides $\mathcal{A} = \{A, C, G, T\}$, a *DNA read* of length $L \in \mathbb{N}^*$ is a sequence $\mathbf{x} = x_1 \dots x_L \in \mathcal{A}^L$. Depending on the sequencing technology, L is typically in the range $50 \sim 400$, and we fix $L = 200$ in the experiments below. For any $1 \leq a \leq b \leq L$ we denote by $\mathbf{x}_{[a,b]} = x_a x_{a+1} \dots x_b$ the substring of $\mathbf{x}$ from position a to b. For any $d \in \mathbb{N}^*$, an *embedding* of DNA reads to $\mathbb{R}^d$ is a mapping $\Phi : \mathcal{A}^L \to \mathbb{R}^d$ to represent each read $\mathbf{x} \in \mathcal{A}^L$ by a vector $\Phi(\mathbf{x}) \in \mathbb{R}^d$, which can then be used for subsequent classification tasks.

For a given $k \in \mathbb{N}$, the k-spectral embedding represents a sequence by its k-mer profile Leslie et al. [2002]: it is an embedding $\Phi^{Spectral} = \left(\Phi_u^{Spectral}\right)_{u \in \mathcal{A}^k}$ in $d = |\mathcal{A}|^k$ dimensions indexed by all strings of length k, where for any such string $u \in \mathcal{A}^k$ one defines:

$$\Phi_u^{Spectral}(\mathbf{x}) = \sum_{i=1}^{L-k+1} \delta_u\left(\mathbf{x}_{[i,i+k-1]}\right) ,$$

where $\delta_u(v) = 1$ if $u = v$, 0 otherwise. The k-spectral encoding of DNA reads is used in state-of-the-art compositional approaches to assign reads to species with machine learning techniques [McHardy et al., 2007; Parks et al., 2011; Patil et al., 2012; Vervier et al., 2016; Wang et al., 2007].

Given $d \in \mathbb{N}^*$ and a $N \times d$ matrix $M = (M_u)_{u \in \mathcal{A}^k}$ associating a vector $M_u \in \mathbb{R}^d$ to each k-mer $u \in \mathcal{A}^k$, we now consider a d-dimensional embedding Φ^M of DNA reads by summing the vectors associated to the read's k-mers:

$$\forall \mathbf{x} \in \mathcal{A}^L , \quad \Phi^M(\mathbf{x}) = \sum_{i=1}^{L-k+1} M_{\mathbf{x}_{[i,i+k-1]}} .$$

In matrix form, one easily sees that the d-dimensional embedding Φ^M can be obtained from the k-spectral representation by the formula:

$$\forall \mathbf{x} \in \mathcal{A}^L , \quad \Phi^M(\mathbf{x}) = M^\top \Phi^{Spectral}(\mathbf{x}) , \tag{2.1}$$

showing in particular that the k-spectral embedding is a particular case of Φ^M by taking $d = N$ and $M = Id$. Changing M allows to create correlations between k-mers in the embedding space. For example, the (k, m)-mismatch kernel Leslie et al. [2003] also corresponds to an embedding Φ^M with $d = N$, but where $M_{u,v} = 1$ when the Hamming distance between u and v is at most m, 0 otherwise. Changing d further allows to vary the dimension of the embedding, which can not only be beneficial for memory and computational reasons, but also help statistical inference by reducing the number of parameters of the embedding.

2.2.2 Learning the embedding

While several existing embeddings such as the k-spectral or (k, m)-mismatch embeddings correspond to Φ^M for specific matrices M, we propose to "learn" M as part of the overall classification or regression task that must be solved. In our metagenomics problem, this is a multiclass classification problem where each of the T bacterial species is a class and each read must be assigned to a class. Given an embedding Φ^M, we consider a linear model of the form:

$$\forall \mathbf{x} \in \mathcal{A}^L\,, \quad f^{M,W}(\mathbf{x}) = W\Phi^M(\mathbf{x})\,, \tag{2.2}$$

where $W \in \mathbb{R}^{T\times d}$ is a matrix of weights, and the prediction rule:

$$\forall \mathbf{x} \in \mathcal{A}^L\,, \quad \hat{y} \in \arg\max_{i\in\{1,\dots,T\}} f_i(\mathbf{x})\,.$$

To learn the embedding M and the linear model M, we assume given a training set of examples $(\mathbf{x}_i, y_i)_{i=1,\dots,n_{train}}$ where $x_i \in \mathcal{A}^L$ and $y_i \in \{1, \dots, T\}$, and numerically minimize an empirical risk:

$$\min_{M,W} \frac{1}{n_{train}} \sum_{i=1}^{n_{train}} \ell\left(y_i, f^{M,W}(\mathbf{x}_i)\right)\,, \tag{2.3}$$

where for the loss ℓ we choose the standard cross-entropy loss after transforming the scores to probabilities with the softmax function:

$$\forall (y, g) \in \{1, \dots, T\} \times \mathbb{R}^T\,, \quad \ell(y, g) = -g_y + \ln\left(\sum_{i=1}^{T} e^{g_i}\right)\,.$$

We solve (2.3) by stochastic gradient descent (SGD). Note that when $d < T$, the problem is usually non-convex and SGD may only converge to a local optimum.

Combining (2.1) and (2.2), we further notice that for any embedding M and weights W,

$$\forall \mathbf{x} \in \mathcal{A}^L\,, \quad f^{M,W}(\mathbf{x}) = WM^\top\Phi^{Spectral}(\mathbf{x})\,. \tag{2.4}$$

This clarifies that $f^{M,W}$ boils down to a linear model in the k-spectral representation, with a weight matrix $WM^\top$ of rank at most d. When $d < \min(N, T)$ this creates a low-rank regularization that can be beneficial for statistical inference, in addition to reducing the memory footprint of the model and speeding up the prediction time.

2.2.3 Implementation

We implemented the model (2.3) by modifying the `fastText` open-source library [Joulin et al., 2016, 2017], which involves a similar model with k-mer embedding for natural language. There are some important differences between DNA reads and standard NLP applications, though: (i) One is that the concept of a "word", space-delimited groups of letters, does not exist in DNA sequence data. Hence we resort to a distributed representation of overlapping k-mers only, and not to words as in `word2vec` or `fastText`. (ii) Second is the number of training examples to be seen by the model is very large: up to 5×10^9 if we were to achieve full coverage of the *large* database below, for example. (iii) Third, the vocabulary is different, as it is of known size (4^k) and is densely represented for relatively small values of k. For greater values of k than those considered in this paper, k-mers become rare and individual long k-mers can become discriminative. This is in fact used by some other compositional algorithms such as Kraken Wood and Salzberg [2014].

For these reasons we rewrote part of the `fastText` software to extract overlapping k-mers rather than words. As appropriate for (iii), the k-mer embeddings are stored in a fixed-size table of dimension $(4^k, d)$, each row corresponding to the vector of a different k-mer. For a given k-mer $\mathbf{b} = b_1 \dots b_k \in \mathcal{A}^k$, the index of its corresponding row in M is $ind(b) = \sum_{i=0}^{k-1} 4^i h(b_i)$, where $h(A) = 0$, $h(C) = 1$, $h(G) = 2$ and $h(T) = 3$. This allows to quickly find $ind(\mathbf{x}_{i+1:i+k})$ from $ind(i : i + k - 1)$ as explained for example in Vervier et al. [2016]. To address (ii) we generate random fragments on the fly directly from the full genomes, rather than reading text samples line by line as in `fastText`.

2.2.4 Regularization with noise

As a form of regularization, we also add random mutations in the training fragments. When reading the fragment from the reference genome, nucleotide by nucleotide, we introduce a chance r of replacing the nucleotide by a random one (equally distributed on $\{A, C, G, T\}$). This is akin to the dataset augmentations commonly used in image classification tasks, and in our case promotes the nearby embeddings of k-mers similar in terms of Hamming distance.

2.3 Experiments

2.3.1 Data

We test our model on two benchmarks proposed by [Vervier et al., 2016]: a `small` one, useful mostly for parameter tuning, and a `large` one. Both benchmarks involve a *training* database of genomes organized by species, and a *validation* set of genomes coming from the same species as the training database, but from different strains. Reads are randomly sampled, with or without noise, from the validation set of genomes, and the goal is to predict, for each read, from which species it comes from. The `small` database contains 356 complete genomes, belonging to 51 species of bacteria; its validation set is composed of 52 genomes, belonging to the same 51 species. The `large` database contains 2,961 genomes belonging to 774 species, which is closer to real-life situations. The validation set is composed of 193 genomes, each from a separate species.

For both training and testing, reads of length $L = 200$ are extracted from the genomes. The validation datasets are built by extracting fragments such that their coverage – the average

number of times each nucleotide is present – is 1. This amounts to a total of 134,319 validation samples for the *small* database, and $\sim$ 3.5M samples for the *large* database. The machine learning-based models are trained on reads sampled from the reference genomes and their known taxonomic labels, while alignment-based methods simply align validation reads to the training reference genomes. To account for the fact that DNA is double-stranded and that when a read is sequenced it can come from any of the two strands, which are reverse-complement to each other, we systematically add the reverse-complement of each read with the same label at training time.

In addition, we consider several *noisy* validation sets as in Vervier et al. [2016], where each fragment sampled from a genome is modified to mimick sequencing errors of actual sequencing machines, in particular substitutions, insertions and deletions of nucleotides. We use the specially-designed `grinder` software [Angly et al., 2012] to simulate 3 new sets of validation reads. The *Balzer* validation set is simulated with a homopolymeric error model, designed to emulate the Roche 454 technology [Balzer et al., 2010]. The *mutation-2* and *mutation-5* sets are simulated with the 4th degree polynomial proposed by Korbel et al. [2009] to study general mutations (insertion/deletions and substitutions). The median mutation rates for these simulated reads are 2% and 5%, respectively. *Balzer* and *mutation-2* are meant to contain a realistic proportion of errors, and *mutation-5* is added as a more challenging set.

2.3.2 Reference methods

We compare our method, which we call `fastDNA` in the rest of the text, to two other strategies. One is the BWA-MEM sequence aligner [Li, 2013] and the other is the linear SVM classifier on the k-spectral representation, implemented using the Vowpal Wabbit software in [Vervier et al., 2016]. We name the latter method VW in the rest of the paper. We follow exactly the same configurations as [Vervier et al., 2016] for both methods.

2.3.3 Small dataset

Memory footprint

As the embeddings matrix M is loaded in memory both for training and classification, `fastDNA` models have a significant memory footprint. [Joulin et al., 2016] discuss various strategies to reduce it. The vocabulary cannot be pruned for our range of k as the k-mers are densely distributed, so the dimensions of M are fixed. The size of M in memory can however be reduced by quantization. We use Product Quantization (PQ, Jegou et al. [2011]), with the option to cluster separately the vector norms and directions (option `qnorm`). The linear layer is then retrained to account for the change in the embeddings. This compresses the model size by almost an order of magnitude without noticeably impacting the performance.

While PQ can make deploying and classifying more accessible, training the model still requires the full embedding matrix, as its quantized version is not trainable. We restrain our parameter choices (k and d) to models that fit on 64GB machines. The dimension of the embedding table (4^k) is encoded on 32-bits, which further limits the value of k to $k_{\max} = 15$. The largest models we consider are $k = 13, d = 100$ and $k = 15, d = 10$. Their memory footprints are available in figure 2.1.

Contrary to VW , model size is completely determined by the user and does not depend on the number of possible classes T or on the vocabulary of the training database, which can

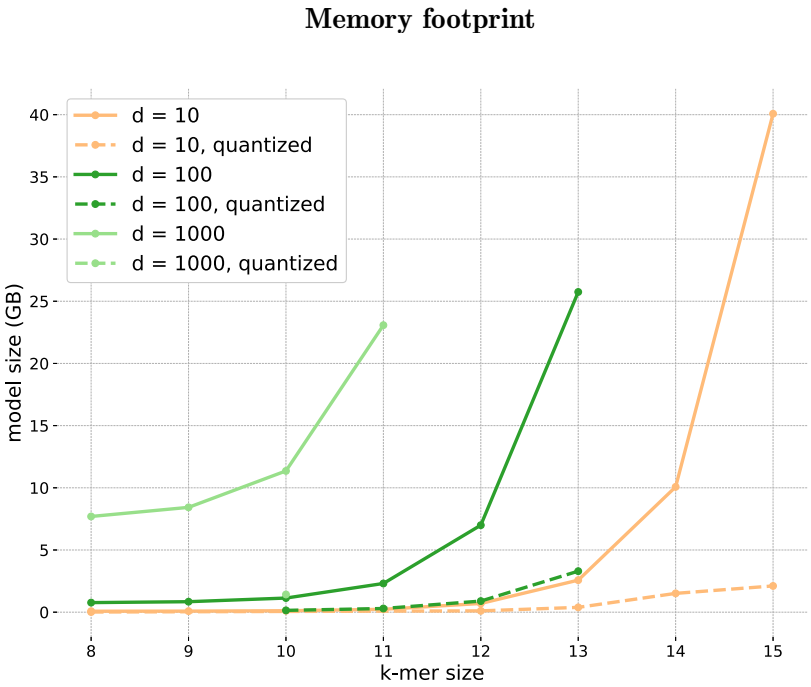


Figure 2.1 – Memory requirement of `fastDNA` models as a function of k-mer sizes. The embedding dimensions d shown are 10, 100 and 1000. The reported value is the size in GB of the model binary, the minimal size required both in RAM to train and load the model, and on disk to save it.

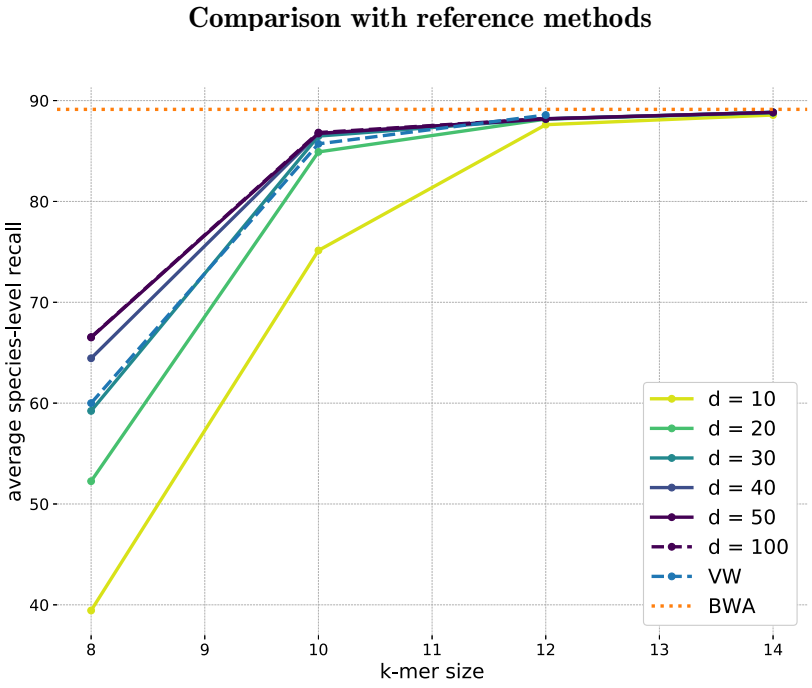


Figure 2.2 – Comparison between `fastDNA` and reference methods on the *small* dataset. This figure shows the average species-level recall obtained by `fastDNA` trained for 50 epochs and a learning rate 0.1, for different values of k and d. The results are compared with VW for different values of k and an alignment-based approach BWA-MEM

become an advantage when T is large.

Coverage

The model is trained by picking a position at random in the reference genomes and using the 200-bp read starting from that position as a sample. One epoch of training consists of drawing enough random reads to cover each nucleotide of the reference genomes once on average (coverage of 1). We found that models with no training noise had converged by 50 epochs, and those with training noise benefited from extra epochs.

The results in this paper were obtained by training with a fixed number of epochs 50, and a learning rate of 0.1, chosen by a standard grid-search.

Performance

One first result from 2.2, is that increasing the embedding dimension d above 50 brings no added benefit in classification quality, at the cost of larger prediction times and memory footprint. This could in theory be expected from (2.4). Once d is greater than the number of classes, the matrix $WM^\top$ has maximal rank T, so the model is virtually the same as the standard "Bag of Words" model VW . The differences observed between the two are likely due to different optimization procedures and implementations.

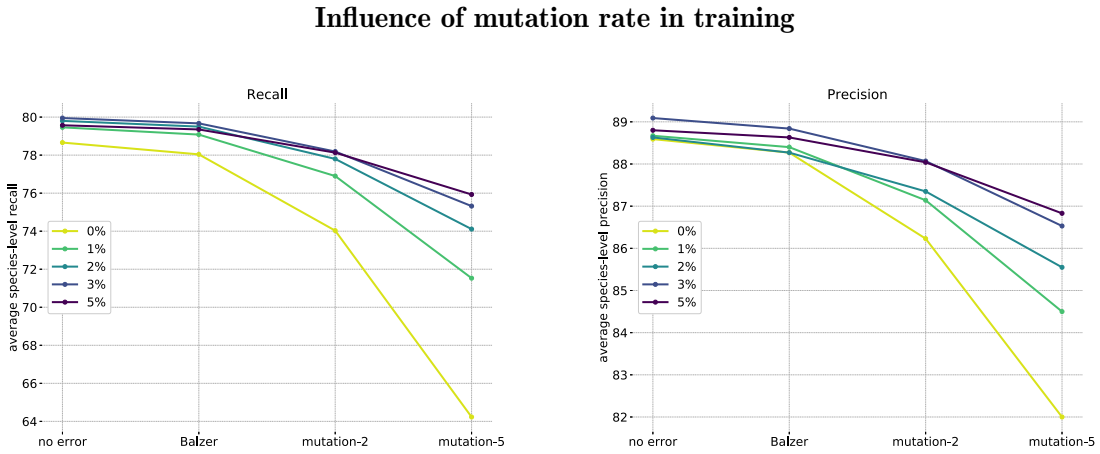


Figure 2.3 – Performance on the large dataset of `fastDNA` trained with different mutation rates. k-mer size and embedding dimension d are 13 and 100, respectively. The classification quality is measured on test sets generated with different sequencing error models.

Excessively lowering the dimension d under T harms the performance, especially for shorter k-mers. However, the gap between models $d = 10$ and models $d = 50$ vanishes as k increases, suggesting that – provided a sufficient vocabulary size – projecting k-mers to a lower-dimensional space comes at little cost. Furthermore, models with longer k-mers but smaller dimension can achieve the same performance as a model with shorter k and greater dimension. The model $k = 14, d = 10$ has the same performance as $k = 12, d = 50$.

Finally, classification performance for values of k greater than 12 is competitive with alignment-based method BWA, which confirms machine learning approaches can be relevant for this problem of taxonomic binning.

2.3.4 Large dataset

We report the classification performance, measured by average species-level recall and precision of `fastDNA` on the validation sets described in 2.3.1. The influence of the training noise is shown in figure 2.3.

As could be expected, greater levels of sequencing noise in the validation sets lead to degraded performances. Adding random mutations to the training reads curbs this effect. The greater the mutation rate is, the more robust the model becomes to higher levels of sequencing noise. Somewhat more surprisingly, a certain range of mutation rates also increases the performance on the validation reads with no sequencing noise. The regularization induced by these artificial errors is therefore beneficial for both sequencing noise and intra-species heterogeneity. We found that the best rates of mutation were between 2 and 5%. The models trained with these levels of noise are better all-around than their no-noise counterparts.

We compare the performance of `fastDNA` against that of VW and BWA in 2.4. For small levels of sequencing errors, `fastDNA` is competitive with BWA. Greater levels in sequencing noise widens the gap between the two, as BWA is very robust to sequencing noise, dropping less than 1% for *mutation-5* .

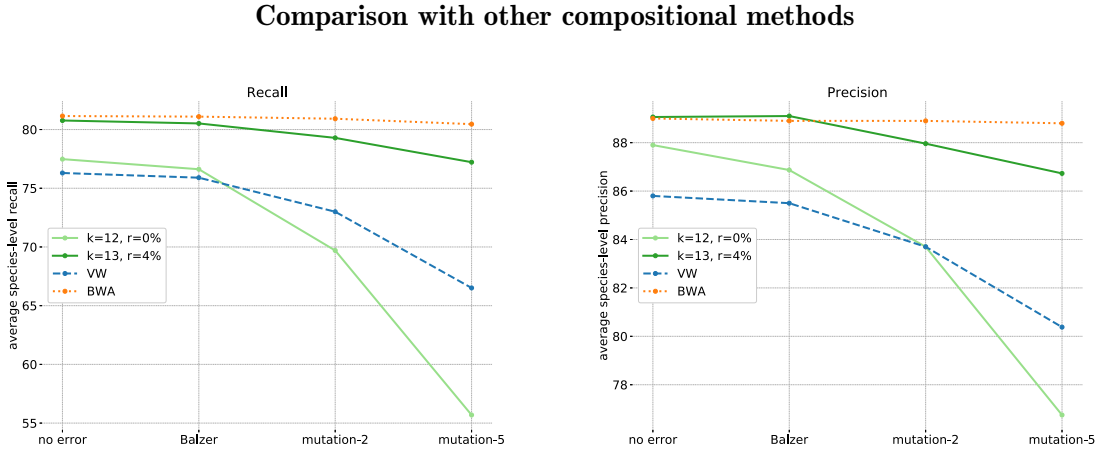


Figure 2.4 – Comparison between `fastDNA` and reference methods on the *large* dataset. This figure shows the average species-level recall and precision obtained by `fastDNA`, VW and BWA on the different validation sets. We show both the best configuration for `fastDNA` ($k = 13$, $d = 100$, $r = 4\%$) and a similar one to VW ($k = 12$, $d = 100$, $r = 0\%$) for a fair comparison.

2.3.5 Classification speed

Speed is of critical importance in taxonomic binning and is the main motivation behind exploring machine learning techniques. Classifying a read with `fastDNA` can be separated in two parts. It first reads the sequence, computes the indices of the k-mers contained in the read and computes the read embedding by summing the k-mer embeddings. This step is of complexity $\mathcal{O}(dL)$, where L is the read length (constant in our experiments). Second, class probabilities are computed by applying the linear classifier, this step is of complexity $\mathcal{O}(dT)$. `fastText` and `fastDNA` offer a different loss function, the hierarchical softmax, that reduces this step to $\mathcal{O}(d \log(T))$, which can become useful in the case of very large T. To this per-read time-complexity must be added a fixed overhead, the time necessary for the model to be loaded from disk, of complexity $\mathcal{O}(dN + T)$, where $N = 4^k$ is the vocabulary size. Due to the large values of 4^k, this can make up a significant portion of the total time. Moreover, we observe in practice a longer memory access time for larger vectors. These are the two reasons for the time gap observed between classification with models of same embedding dimension d but different k.

The total time complexity for predicting a dataset with n samples is therefore $\mathcal{O}(n(dL + dT) + dN)$.

With reads of constant length, `fastDNA` and VW classify reads indiscriminately of their content, and therefore yield equal classification times across the validation sets. On the other hand, BWA's speed degrades with the sequencing noise, which is easily explained. BWA searches iteratively on the number of mismatches z and stops once it gets a hit. It will therefore be slower if there are more mismatches between the testing and reference data.

We show in figure 2.5 the classification speeds measured for the *large* dataset. The values reported are for a single CPU (Intel Xeon E5-2450 v2 -2.5GHz). We show `fastDNA` for $d = 100$ and $k = 12$, 13 and VW . `fastDNA` and VW have similar classification times of $\sim 6,7 \cdot 10^3$ reads per minute. As remarked in Vervier et al. [2016], compositional approaches offer systematically better prediction times than BWA, with improvements of $2 - 9\times$. This speed improvement

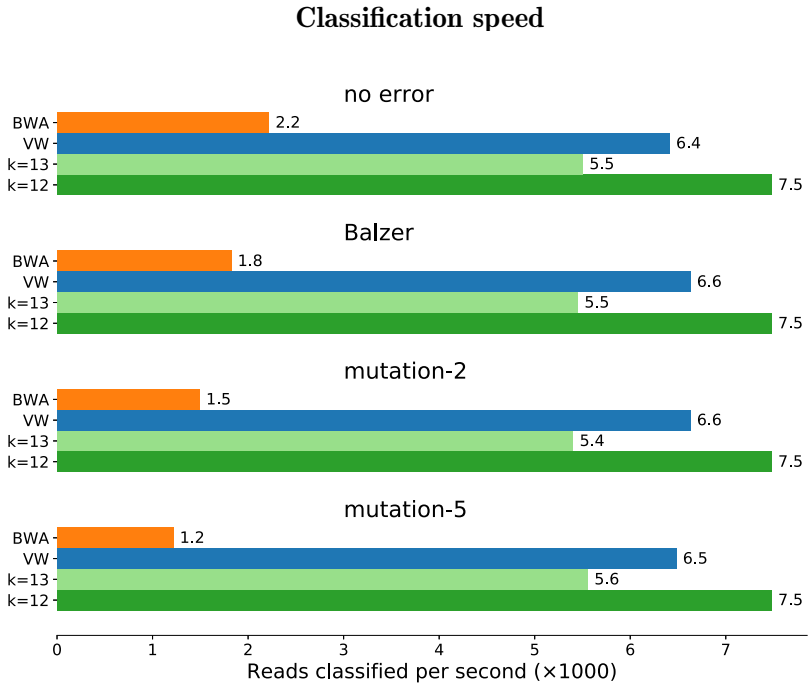


Figure 2.5 – Comparison between `fastDNA` and reference methods on the *large* dataset. This figure shows the average classification speed of the methods on the different test sets. Two versions of `fastDNA` are shown, one with k-mer size 12, the other with k-mer size 13. Both have embedding dimension $d = 100$. The four test sets used were simulated with different sequencing error models.

increases with the mismatch between predicted sequences and reference genomes, therefore with both sequencing noise and intra-species variations.

2.4 Conclusion

We demonstrated that learning a low-dimensional representation of DNA reads based on their k-mer composition is feasible, and outperforms state-of-the-art compositional approaches that work directly on the high-dimensional, k-spectral representation of DNA sequences. Controlling the dimension d of the embedding allows to consider longer k-mers for a given memory footprint. As other compositional methods, `fastDNA` is significantly faster than alignment-based methods, and is well adapted to classification into many classes.

There are two immediate possible extensions of this work. One is to use a more realistic error model than uniform substitutions for training, to better mimick the expected noise in the data. The other is to extend the notion of "word" from contiguous k-mers to gap-seeded k-mers or Bloom filters Luo et al. [2017], hopefully capturing longer-range dependencies. In terms of applications, assigning RNA-seq reads to genes to quantify their expression can also be formulated as a classification problem with typically $T \sim 22k$ classes, and may be well adapted to `fastDNA` as well.

The source code is freely available and published on github `https://github.com/rmenegaux/`

fastDNA, along with scripts to reproduce the presented results.

Chapter 3

Brume: Embedding the de Bruijn graph

Contents

3.1 Introduction . . . 46
3.2 Approach . . . 47
3.2.1 Two-layer classifier . . . 47
3.2.2 Exploiting k-mer symmetry . . . 48
3.2.3 Exploiting k-mers binning with the de Bruijn graph . . . 48
3.2.4 Implementation . . . 51
3.3 Methods . . . 52
3.3.1 Data . . . 52
3.3.2 Reference methods . . . 52
3.4 Results . . . 52
3.4.1 Embedding canonical k-mers . . . 52
3.4.2 Embedding contigs . . . 53
3.5 Discussion . . . 57
3.6 Conclusion . . . 58

Abstract

Motivation: Fast mapping of sequencing reads to taxonomic clades is a crucial step in metagenomics, which however raises computational challenges as the numbers of reads and of taxonomic clades increases. Besides alignment-based methods, which are accurate but computational costly, faster compositional approaches have recently been proposed to predict the taxonomic clade of a read based on the set of k-mers it contains. Machine learning-based compositional approaches, in particular, have recently reached accuracies similar to alignment-based models, while being considerably faster. It has been observed that the accuracy of these models increases with the length k of the k-mers they use, however existing methods are limited to handle k-mers of lengths up to $k = 12$ or 13 because of their large memory footprint needed to store the model coefficients for each possible k-mer.
Results: In order to explore the performance of machine learning-based compositional approaches for longer k-mers than currently possible, we propose to reduce the memory footprint of these methods by binning together k-mers that appear together in the sequencing reads used to train the models. We achieve this binning by learning a vector embedding for the vertices of a compacted de Bruijn graph, allowing us to embed any DNA sequence in a low-dimensional vector space where a machine learning system can be trained. The resulting method, which we call `Brume`, allows us to train compositional machine learning-based models with k-mers of length up to $k = 31$. We show on two metagenomics benchmark that `Brume` reaches better performance than previously achieved, thanks to the use of longer k-mers.
Availability: The source code is freely available and published on github https://github.com/rmenegaux/fastDNA, on the `kallisto` branch, along with scripts to reproduce the presented results.
Contact: jpvert@google.com romain.menegaux@mines-paristech.fr

Résumé

Motivation : Une étape cruciale en métagénomique est d'attribuer un clade taxonomique aux fragments d'ADN issus du séquençage. Avec l'augmentation constante du volume de données, cette tâche pose des problèmes computationnels. Au-delà des méthodes basées sur l'alignement, précises mais coûteuses, des approches compositionnelles plus rapides ont récemment été proposées, prédisant le clade taxonomique d'un fragment d'ADN à partir des k-mers qui le composent. Les approches basées sur l'apprentissage statistique ont notamment atteint une précision comparable aux méthodes basées sur l'alignement, tout en étant considérablement plus rapides. Il a été montré empiriquement que la précision de ces modèles augmente avec la longueur des k-mers, cependant les méthodes existantes sont limitées à une longueur de $k = 12$ ou 13 à cause de la grande quantité de mémoire requise pour stocker les paramètres de chaque k-mer.
Résultats : Afin d'explorer la performance d'approches compositionelles statistiques pour des k-mers plus longs, nous proposons de réduire l'impact en mémoire de ces méthodes en regroupant les k-mers apparaissant dans les mêmes fragments des données d'entraînement. Nous y parvenons en apprenant une représentation vectorielle pour chaque noeud d'un graphe de Bruijn compacté, ce qui permet de donner une représentation vectorielle de basse dimension pour chaque fragment d'ADN et ainsi d'entraîner un système d'apprentissage statistique. La méthode, que nous nommons `Brume`, permet d'entraîner des modèles d'apprentissage statistique avec des longueurs de k-mers allant jusqu'à $k = 31$. Nous montrons sur deux jeux de données de métagénomique que `Brume` atteint une meilleure performance que les approches précédentes, grâce à ces longs k-mers.
Disponibilité : Le code source est disponible sur le github `https://github.com/rmenegaux/fastDNA`, sur la branche `kallisto`, accompagné de scripts pour reproduire les résultats présentés.
Contact : jpvert@google.com romain.menegaux@mines-paristech.fr

3.1 Introduction

The cost of DNA sequencing has reduced dramatically over the past years. Among its many applications, it has become the method of choice for metagenomics, a field that aims at characterizing an environment directly from the DNA it contains by sequencing DNA samples randomly collected from the environment. A bottleneck in modern metagenomics pipelines is the taxonomic binning step, i.e., the mapping of the output of second generation shotgun sequencing machines – millions to billions of short DNA reads – to known taxonomic clades. Of the several methods developed to tackle this problem, a natural one is to use an all-purpose DNA aligner, such as Li [2013], Langmead et al. [2009a] or Li [2018], to align the reads to reference genomes. These aligners, which are usually based on string-matching algorithms, are very accurate but typically computationally too slow for taxonomic binning. More recently, competitive performances have been achieved by so-called compositional approaches, in terms of both speed and precision. Compositional approaches do not try to align a candidate read to a reference genome, but instead directly predict the genome the read is likely to come from based on the composition of the read in shorter strings, called k-mers. These methods can be roughly separated in two separate classes. Methods of the first group, so-called pseudo-alignment methods [Ounit et al., 2015; Wood and Salzberg, 2014; Wood et al., 2019], break a read into long and hopefully characteristic subsequences (k-mers, $k \sim 30$) and match the read to the genomes/species that has the most k-mer matches. Usually a single match on those long k-mers is enough to ensure classification to a taxonomic clade. The second class of compositional methods treat the problem as a machine learning classification problem. Most of these approaches represent the DNA read as a set of short k-mers (k from 4 to 14) and then use a discriminative machine learning model to predict the clade of a read from the vector of frequencies of all k-mers it contains. In Vervier et al. [2016], this model is simply linear, akin to the bag-of-words model in natural language processing, and k-mers of lengths up to $k = 12$ are considered. Menegaux and Vert [2019] use a two-layer model called `fastDNA`, which first embeds the k-mers in a continuous low-dimensional vector space and then classifies reads in the vector space with multinomial logistic regression. The low-dimensional embedding in Menegaux and Vert [2019] allows them to reduce the memory footprint of the model when many possible taxonomic clades exist, and to consider k-mers of length up to $k = 13$. More recently deep learning has also been applied with some success with de DeepMicrobes model of Liang et al. [2019], by adding a recurrent neural network on top of these embeddings; Liang et al. [2019] consider k-mers of length $k = 12$, the largest possible that allows to fit the model in the memory of the hardware needed to train the model.

Results from Liang et al. [2019]; Menegaux and Vert [2019]; Vervier et al. [2016] all suggest that k-mers with larger k lead to better performance. Storing the parameters of the models in memory is however often the limiting factor to increase k, since the memory footprint of the model is typically $O(c4^k)$ when a model is stored for each of c clades [Vervier et al., 2016], or $O(d4^k)$ when the k-mers are first represented in a d-dimensional vector space Liang et al. [2019]; Menegaux and Vert [2019]. The exponential growth of the memory footprint with k has thus far limited the use of machine learning-based methods to $k = 12$ or 13. Pseudo-alignment methods, on the other hand, work with longer discriminative k-mers (k between 20 and 31), but then use a very simple presence/absence model to classify the reads, and do not need to store any parameter for all possible k-mers.

In this work, we propose a new approach to reduce of memory footprint of machine learning-

based models for large k, which allows us to explore the performance of these models for k up to 31. Our approach is based on binning together k-mers which occur consecutively in the same reads in the training set, and learning a single vector representation for each such bin. These bins, also called contigs or unitigs, correspond to vertices in a compact de Bruijn graph of the training sequences. Instead of storing vector representations for 4^k k-mers, our approach thus reduces the need to store representations for only n_c contigs, which can be much smaller than 4^k.

We implement this idea in a software, `Brume`, which extends the two-layer model of Menegaux and Vert [2019] by embedding contigs instead of k-mers in the first layer. We report promising results of `Brume` on two metagenomics benchmarks, where increasing k beyond 12 or 13 leads not only to better performance compared to smaller k, but also compared to alignment-based approaches.

3.2 Approach

3.2.1 Two-layer classifier

Here we describe the two-layer machine learning model for read classification deployed in `fastDNA` [Menegaux and Vert, 2019], which we generalize below. We consider a finite alphabet $\mathcal{A}$, which for our purpose is $\mathcal{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, the set of four nucleotides found in DNA. A k-mer $x \in \mathcal{A}^k$ is a string of fixed length k. In `fastDNA`, a d-dimensional embedding is first learned for each k-mer. If $N = 4^k$ is the total number of possible k-mers, the embeddings matrix M is a $N \times d$ matrix with each row M_s coding the vector representation of a k-mer s. From the embedding matrix M of all k-mers, we then define the embedding $\Phi^M(\mathbf{x}) \in \mathbb{R}^d$ of a read $\mathbf{x} \in \mathcal{A}^L$ (for $L \geq k$) as the average embedding of its constituent k-mers, i.e.:

$$\forall \mathbf{x} \in \mathcal{A}^L\,, \quad \Phi^M(\mathbf{x}) = \frac{1}{L-k+1} \sum_{i=1}^{L-k+1} M_{\mathbf{x}_{[i,i+k-1]}}\,. \tag{3.1}$$

Given the embedding Φ^M (corresponding to the first layer of the model), we consider a linear model (second layer) of the form:

$$\forall \mathbf{x} \in \mathcal{A}^L\,, \quad f^{M,W}(\mathbf{x}) = W\Phi^M(\mathbf{x})\,, \tag{3.2}$$

where $W \in \mathbb{R}^{T \times d}$ is a matrix of weights and T is the number of taxonomic clades, and the prediction rule:

$$\forall \mathbf{x} \in \mathcal{A}^L\,, \quad \hat{y} \in \arg\max_{i \in \{1,\dots,T\}} f_i(\mathbf{x})\,.$$

The two-layer classifier is thus parameterized by the two matrices M and W. Given a training set of reads with known taxonomic clades, which are typically computationally generated by randomly sampling DNA fragments from known organisms and, potentially, adding noise to the fragments, the two matrices are optimized by minimizing an empirical error such as the mean cross-entropy loss over the training set using a standard first-order optimization algorithm such as stochastic gradient descent.

3.2.2 Exploiting k-mer symmetry

A limitation of the two-layer classifier is that the size in memory of the representation matrix M (with $4^k \times d$ entries) can quickly become prohibitive as k, and the number of k-mers, grows. Current implementations of `fastDNA` and similar methods such as DeepMicrobes use for example $k = 13$ and $k = 12$ respectively. On the other hand, it was reported in Menegaux and Vert [2019]; Vervier et al. [2016] that increasing k generally improves the accuracy of the models. We therefore look for strategies to reduce the memory footprint of M, in order to explore models with larger k's.

The idea we pursue in this paper is to bin k-mers into $N < 4^k$ groups, and to enforce all k-mers in any given group to have the same vector representation. Computationally, this allows to reduce the memory footprint of the model by storing only a $N \times d$ matrix of vector representations for the groups, in addition to a lookup table or function to quickly map each k-mer to its group.

A first idea to roughly halve the memory footprint of the embedding matrix is to exploit the natural symmetry of DNA, and impose that a read and its reverse complement are indistinguishable in the representation space. The same then goes for k-mers: a k-mer and its reverse complement should have the same representation. Hence a first step is to learn an embedding per canonical k-mer, which roughly halves the number of embeddings. Formally, if for a k-mer x we denote its reverse complement by $\bar{x}$, then the canonical k-mer $\hat{x}$ is the smallest of x and $\bar{x}$ in the lexicographic order. We detail in the Appendix how the lookup mapping from a k-mer to its canonical form is performed, and implemented this modification in the vanilla `fastDNA` software.

3.2.3 Exploiting k-mers binning with the de Bruijn graph

Exploiting k-mer symmetries can roughly halve the memory footprint of the representation matrix M, which is not enough to really increase k since each increase of k by 1 multiplies the number of k-mers by 4. Here we propose another, more drastic grouping of k-mers by exploiting the idea that k-mers that always appear together in the same reads contain the same information for classification purposes, hence should have the same embedding. Indeed, due to the form of embedding for a read (3.1), one can see that if several k-mers are always present in or absent together from a set of reads, then their contribution to the embedding of any read in the set will always be either zero if they are absent from the read, or a constant equal to the sum of their individual embeddings if they are present in the read. Applying this observation to the reads used to train the two-layer classifier, we conclude that the only information that can be used to optimize the k-mers representation is at the level of k-mers groups. In other words, instead of coding an individual embedding for each k-mer, one can get the same modelling capacity by just coding an embedding for that set of k-mers, that would correspond to their constant contribution to a read embedding when they are present in the read.

In order to turn this idea into practice, we need a way to bin k-mers into groups such that k-mers in a group are always present or absent together in the reads used to train the model. For that purpose we exploit the notion of *de Bruijn graph* (dBG), which we now recall. The dBG of a set of sequences S is a directed graph (V, E) where the vertices V are the k-mers appearing in S. There is an edge $e = (k_1, k_2)$ between two vertices if and only if their corresponding k-mers are adjacent in one of the sequences. In the colored de Bruijn graph (cdBG), edges are triplets (k_1, k_2, c) where the color c codes the sequence in which k_1 and k_2 are neighbors. A maximal

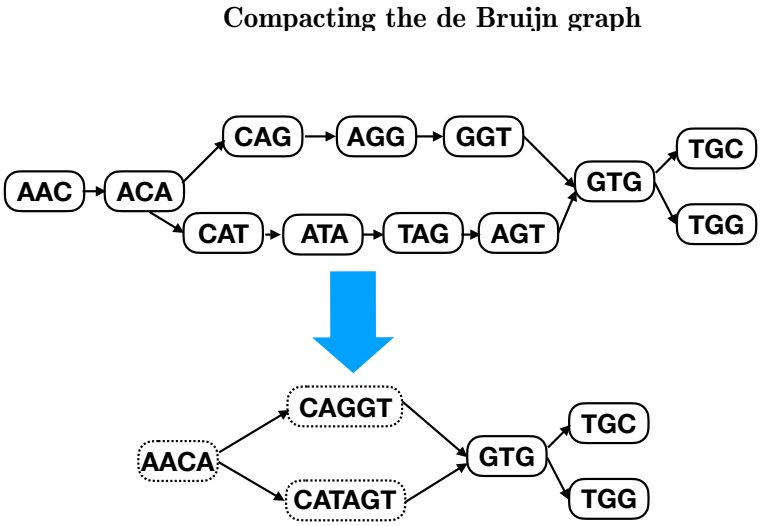


Figure 3.1 – Above: de Bruijn graph for the two sequences `AACAGGTGC` and `AACATAGTGG`. Below: corresponding compacted de Bruijn graph. Each node in this graph is a *contig*.

non-branching path in a dBG is called a *contig*, or unitig. In the case of a cdBG, a contig C is a set of contiguous k-mers k_i that are linked by edges of the same colors c. Figure 3.1 illustrates the concepts of dBG and cdBG for a set of two sequences.

All of the k-mers appearing in the reference genomes belong to one and only one contig. As a consequence the number of contigs N_C is bounded by the number of k-mers N_k.

We now propose to bin k-mers by the unique contig they belong to in the cdBG of the training set of reads, and only learn a vector representation for each contig. The embedding of a k-mer is then simply the embedding of the contig it belongs to. This reduces the memory footprint of the embedding matrix from $4^k \times d$ to $N_C \times d$, which can be a strong reduction when there are many less contigs than possible k-mers. For example, in Figure 3.2 we see that as soon as k is larger than 15 then the number of contigs flattens and becomes orders of magnitude smaller than the number of possible k-mers, illustrating the potential benefits of the approach.

A caveat with this approach is that k-mers that do not appear in the training set of reads have no associated contig. Such k-mers occur in a read for which we want to make a prediction, and is even likely to occur more frequently when k is large. For example, in Figure 3.2 we see that for $k \geq 15$ there is a gap between k-mers present in the training set and possible k-mers. By default, we ignore k-mers absent from the training set by setting their embedding to zero. This also suggests that combining large and small k-mers in a single model may be beneficial in some cases, a direction we explore empirically below.

Input read R ;
Decompose R into its constituent k-mers $v_1, ..., v_n$;
Lookup each v_i in the dBG and find its parent contig C_i;
Compute the embedding $\Phi(R) = 1/n \sum_{i=1}^{n} \Phi(C_i)$;

Algorithm 1: `Brume` embedding

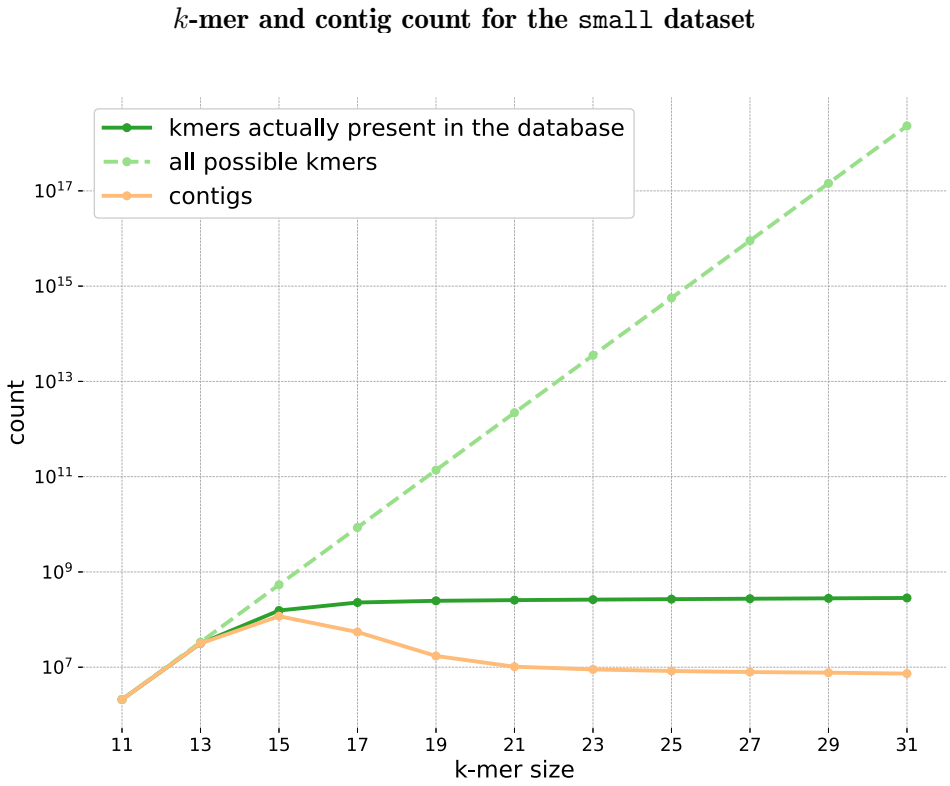


Figure 3.2 – Comparison of the number of k-mers and number of contigs present in the reference genomes of the `small` dataset. The dotted line is the theoretical number of possible canonical k-mers ($= 4^k/2$)

3.2.4 Implementation

We implemented the model as an extension of `fastDNA`. To build the de Bruijn graph, we reused the implementation of `kallisto` [Bray et al., 2016].

3.3 Methods

3.3.1 Data

We test our method on the two datasets proposed in Vervier et al. [2016] and Menegaux and Vert [2019], called `small` and `large`. Both were extracted from the NCBI bacterial database (reference), and both have a training set and a validation set. In `small` (resp. `large`) the training database is composed of 356 (resp. 2961) bacterial genomes, coming from 51 (resp. 774) species. The validation database is 52 (resp. 193) different genomes, coming from the same 51 (resp. 774) species.

For both training and validation, reads of length $L = 200$ are randomly extracted from the reference genomes. We call a training *epoch* a set of reads such that each base-pair appears once on average (coverage of 1). Epochs are generated on the fly directly during training. We perform data augmentation by adding the possibility to inject random mutations to training reads at a fixed rate r, as this was shown to improve performance by Menegaux and Vert [2019]

For validation, we create 4 separate sets from the validation genomes. Each contains about 3.5M (134K for `small`) reads of length $L = 200$, corresponding to a coverage of 1. The first set, called *no-noise*, is extracted as is, with no sequencing noise. The 3 others are extracted with `grinder` Angly et al. [2012], a software specifically designed to mimic sequencing noise from real machines. *Balzer* emulates the Roche 454 technology, and is simulated with a homopolymeric error model [Balzer et al., 2010]. *Mutation-2* and *mutation-5* are simulated with the 4th degree polynomial proposed by Korbel et al. [2009] to study the effect of substitutions, insertions and deletions. The median level of mutations is 2% and 5%, respectively. *Balzer* and *mutation-2* contain error-levels expected from current technology, whereas *mutation-5* is intended to be an extreme case.

We report as metrics the average species-level recall and accuracy, as well as the F1-score to balance between the two.

3.3.2 Reference methods

We compare our results to a standard aligner, BWA-MEM (Li [2013]), that is commonly used in production for metagenomics analysis. We also compare to the standard `fastDNA` method with best parameters from Menegaux and Vert [2019] ($k = 13$, training with mutation noise $r = 4\%$), in order to assess the benefits of increasing k beyond $k = 13$.

3.4 Results

3.4.1 Embedding canonical k-mers

Learning an embedding for each canonical k-mer, rather than for all of them, cuts memory usage by roughly a factor of 2. This enables training a `fastDNA` model with $k = 14$ and $d = 50$, rather than the state of the art $k = 13$, $d = 100$ presented in Menegaux and Vert [2019]. As shown in Figure 3.3, this new model is better in both sensitivity and specificity than the previous best, for all levels of sequencing noise, on the `large` dataset. For reads with little noise, it is also better than the alignment method BWA, the latter's performance being however more robust to sequencing noise. This confirms the findings of Menegaux and Vert [2019]; Vervier et al. [2016]

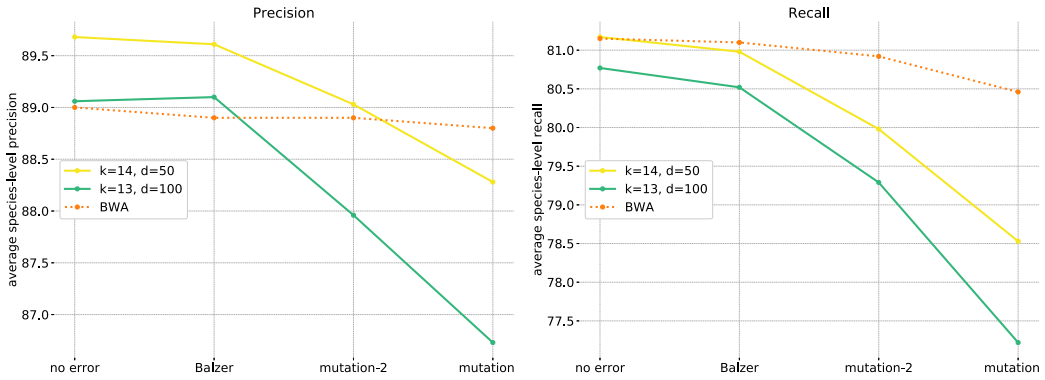


Figure 3.3 – Performance of `fastDNA` with our new indexing (yellow) on the `large` dataset. We compare to the results of the BWA aligner (orange dotted line), and to the state of the art `fastDNA` method, with parameters $k = 13$, dimension $d = 100$ (green). Training mutation rate for both models was $r = 4\%$

that increasing k in machine learning-based compositional models can be beneficial, and lead to model competitive with alignment-based approaches.

3.4.2 Embedding contigs

For the rest of this section, we focus on the proposed algorithm `Brume`, which learns an embedding per contig of the cdBG of the training set in order to reduce the memory footprint of the embedding layer. This allows us to explore machine learning-based compositional models for larger values of k.

Performance on the `small` dataset

Figure 3.4 shows the performance of `Brume` on the `small` dataset as a function of k, with k ranging from 11 to 31. Notice that due to memory constraints, previous work has only investigated values of k up to 12 or 13. Notice also that we stopped at $k = 31$ not for memory reasons, but because the performance of the models do not seem to improve by further increasing k. Finally, note that for k less than 13, there is almost a one to one mapping between k-mers and contigs, so results between `fastDNA` and `Brume` are expected to be identical.

We see in Figure 3.4 that recall is maximum at $k = 15$, which coincides with the maximum number of contigs suggesting that perhaps the shattering dimension is what counts ultimately. Larger k lead systematically to a decrease in recall, and to an increase in precision from $k = 17$ upwards. The F1-score is optimum for $k = 23$, which shows that our approach is promising.

Performance on the `large` dataset

We now turn to the more challenging and realistic `large` datasets. Figure 3.5 shows the performance on the `large` dataset as a function of k. As reference we show the performance of BWA, and the best performance of `fastDNA` achieved by embedding canonical k-mers (achieved

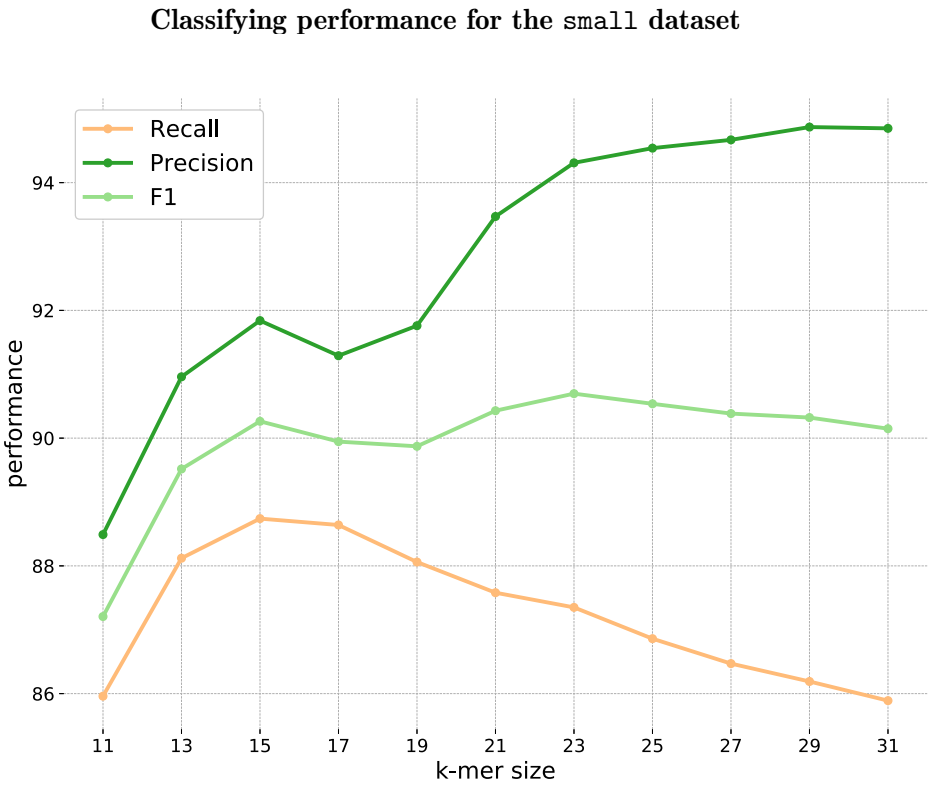


Figure 3.4 – Average species-level precision, recall and F1-score for the `small` dataset, as a function of k-mer length. The model were trained for 50 epochs with an embedding dimension of $d = 10$

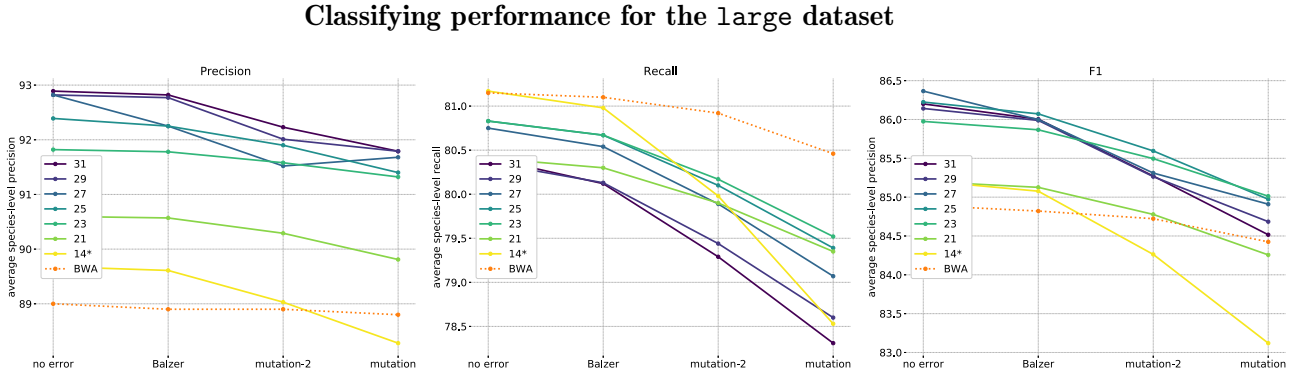


Figure 3.5 – Performance of our method on the `large` dataset, as a function of k-mer length k (ranging from 21 to 31). The embedding dimension is $d = 50$, and the models were trained for 50 epochs. We compare to the results of the BWA aligner (orange dotted line), and to the `fastDNA` method (yellow), with best parameters $k = 14$ and training mutation rate $r = 4\%$)

for $k = 14$, $d = 50$, $r = 4\%$). We do not have values for $k = 17$ and $k = 19$ because the dBGs were too large to be built or to fit in memory.

Overall, we see that, like on the `small` dataset, there is an important benefit in F1 score when k increases. In particular, for k between 23 and 31, the F1 score of `Brume` is above both `fastDNA` and BWA on all four datasets.

An interesting observation is that models with larger k's look generally more robust to sequencing errors . One can see in Figure 3.5 that their performance, in terms of F1, recall and precision, decreases more slowly than that of `fastDNA` with higher levels of noise. A possible explanation is that one mutation in a read leads all k neighboring k-mers to be corrupted. The larger k is, the more specific they are and most of them will not find a match in the cdBG, and therefore will not impact the embedding or the classification. On the contrary, `fastDNA` treats these erroneous k-mers as any other.

Similarly to the *small* dataset, the increase in F1 score for larger k's hides the fact that recall (resp. precision) consistently decreases (resp. increases) with larger k's. The increase in precision for larger k can be at least partly attributed to an increased proportion of reads that are not classified. As seen in figure 3.6, this proportion can go over 10% for the `large` dataset, and increases with both k and the level of noise. Unclassified reads are reads with no matching k-mers found in the dBG of the training set, and the predictions are left blank for them: they count as errors for recall but do not count for precision. The fact that precision goes up as recall goes down suggests that those missing reads are precisely those which are difficult to classify.

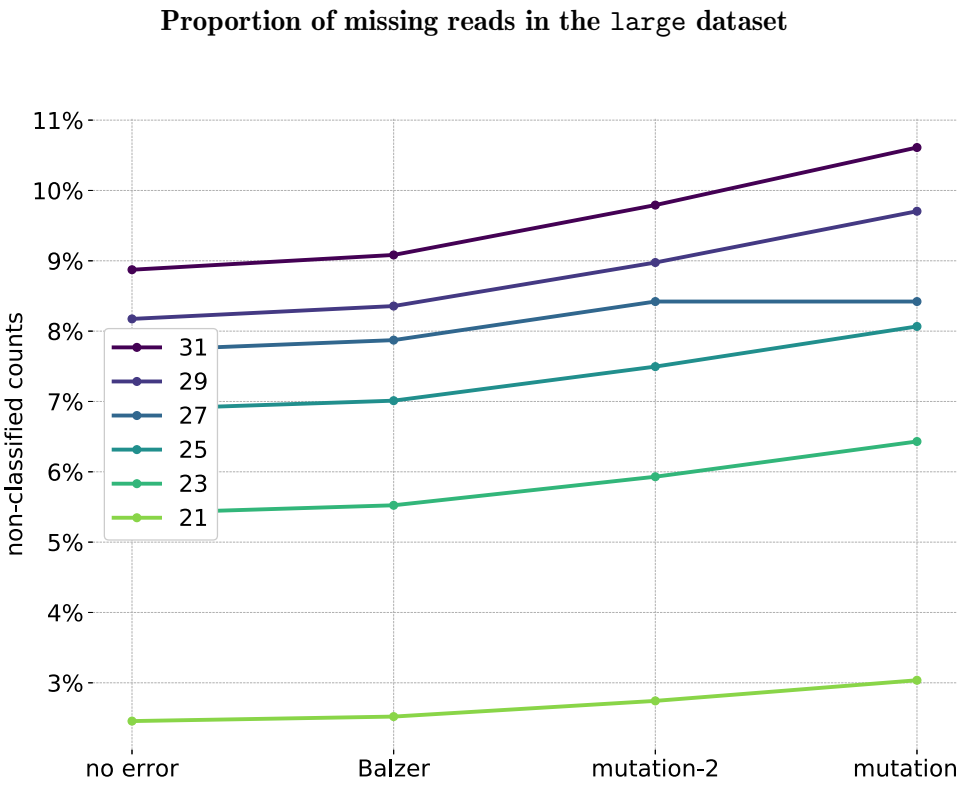


Figure 3.6 – Proportion of validation reads in `large` with no matches in the reference genomes as a function of simulated sequencing noise. Each line is a fixed value of k-mer length k.

3.5 Discussion

We demonstrated that k-mer-based machine learning methods for read classification can be extended to large k, by quantizing the k-mer space. We did so by using the de Bruijn graph and giving the same representation to k-mers that were in the same contig. A supervised classifier, similar in architecture to `fastDNA`, outperforms state of the art in terms of precision and F1-score.

When k gets large, k-mers become increasingly specific to each read and we are increasingly confronted to the situation where a read to be classified has no k-mer in common with the training set used to train the model. In that situation, we do not classify the read, which decreases the recall of our model, but as we observed also increases the precision since those reads tend to be "hard" to classify. In order to try to classify reads that have been left out, we could also make use of smaller, dense k-mers to give them a representation. This naturally leads to the idea of mixing representations with large and small k. We have tried several ways to do so, in particular:

1. Training a separate `fastDNA` model with short k that we use only in case `Brume` cannot match the read

2. Training separate embeddings for short k and long K, then averaging or concatenating them to yield a unified embedding.

We implemented and tested these methods, but overall the classification performance on the homeless reads is not good enough to justify the extra workload. For example, in Figure 3.7, we show the results for a hybrid model $k = 13, K = 31$, which gives a d-dimensional embedding to both short k-mers and contigs, then averages the two to produce the final representation of a read. Recall is indeed boosted but at a high cost in precision.

This shows that combining short and long k-mers as we did is not enough to improve both precision and recall, and suggests that increasing k allows to navigate a trade-off between precision and recall, where models with larger k's simply do not classify "hard" reads and are more accurate on "easy" reads.

Regarding speed, the k-mer lookup in the dBG comes with a significant impact on classification speed. `kallisto` have a heuristic based on the dBG to avoid looking up every k-mer, which we have not reimplemented as of now. We have also experimented with another library to build the dBG called `bifrost` [Holley, 2020], which is slightly faster, presumably because of the rolling hash they use to lookup k-mers. The built dBGs were essentially the same and therefore the classifying performance of `Brume` was unchanged. Results are not shown here.

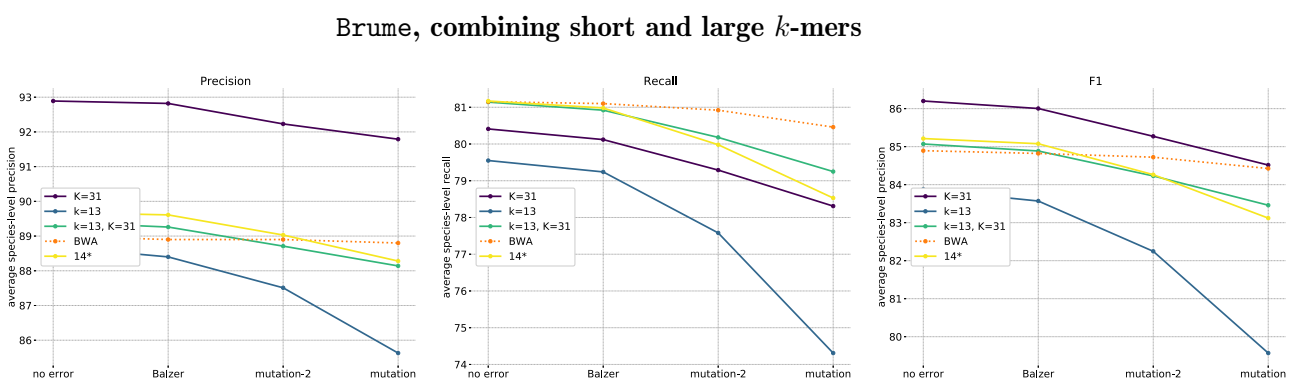


Figure 3.7 – Performance on the `large` dataset of a hybrid model. The embedding of a read is the average of the embeddings of its 13-mers and its 31-mers. Dimension is $d = 50$. Shown as comparison are `Brume` with $k = 31$ and `fastDNA` with $k = 13$.

3.6 Conclusion

We have proposed a new way to bin k-mer representation for machine learning-based models, using a binning based on contigs in a cdBG, and illustrated the benefits of using larger k values than currently available on a metagenomics application. We believe these embeddings can be used for other more sophisticated models, such as deep learning-based models, and in other fields than metagenomics, such as RNA-seq.

Appendix: Indexing scheme

Because of the natural symmetry of DNA, a read and its reverse complement should be indistinguishable in our model. The same goes for k-mers: a k-mer and its reverse complement should have the same representation.

Let $M^k = \{(u, \bar{u}), u \in \mathcal{A}^k, u \leq \bar{u}\}$ the set of alphabetically ordered pairs of k-mers and their reverse complements. Let $m_k = \|M^k\|$. A palindrome is a k-mer u such that $u = \bar{u}$ The number of elements in M^k is half the number of elements in $\mathcal{A}^k$ plus half the number of palindromes, so $m_k = 4^k/2$ if k is odd (no palindromes) and $m_k = 4^k/2 + 2^k$ if k is even.

An indexing scheme is a bijection ϕ from M^k to $[0, m_k - 1]$.

We define ϕ recursively on k as follows:

1. ϕ of the empty string is 0: $\phi('') = 0$
2. ϕ of a single character $a \in \mathcal{A}$ is $\phi(a) = \{\texttt{A} : 0, \texttt{C} : 1, \texttt{G} : 2, \texttt{T} : 3\}[a]$:
3. For $u \in \mathcal{A}^k$, we write $u = u_1 u' u_k$, where $u_1, u_k \in \mathcal{A}$ are the first and last characters of u, then use the rule described in Algorithm 2

$m_k = 4^k/2 + \mathbb{1}_{k \text{ is even}} 2^k$ is the number of canonical k-mers
Indexing scheme ϕ is a function from $\mathcal{A}^k$ to $[0, m_k - 1]$
$\phi('') = 0$;
$\phi(a) = \{\texttt{A}: 0, \texttt{C}: 1, \texttt{G}: 2, \texttt{T}: 3\}[a]$;
Let $u = u_1 u' u_k \in \mathcal{A}^k$, where $u_1, u_k \in \mathcal{A}$ and $u' \in \mathcal{A}^{k-2}$;
if $u_1 u_k \in \{\texttt{AT, TA, CG, GC}\}$ **then**
  $\phi(u) = \{\texttt{AT}:0, \texttt{TA}:1, \texttt{CG}:2, \texttt{GC}: 3\}[u_1 u_k] m_{k-2} + \phi(u')$;
else
  $\phi(u) = 4m_{k-2} + \{\texttt{AA/TT}: 0, \texttt{AC/GT}: 1, \texttt{AG/CT}: 2, \texttt{CA/TG}: 3, \texttt{CC/GG}: 4, \texttt{GA/TC}: 5$
  $\}[u_1 u_k] 4^{k-2} + \phi(u')$
end

Algorithm 2: `fastDNA` symmetric and bijective indexing scheme

Chapter 4

Adapting the hierarchical softmax for taxonomic classification

Contents

4.1 Introduction . . . 63
4.2 Related Work . . . 64
4.2.1 Approximating the softmax . . . 64
4.2.2 Leveraging label taxonomy for classification . . . 65
4.3 Methods . . . 66
4.3.1 Hierarchical softmax . . . 66
4.3.2 Tree construction . . . 67
4.4 Results . . . 68
4.4.1 Learning the input representation . . . 68
4.4.2 Data . . . 69
4.4.3 Classification accuracy . . . 69
4.4.4 Classification speed . . . 69
4.5 Discussion and future work . . . 71

Abstract

We present a new structured loss for the task of assigning taxonomic labels to DNA sequencing reads – the so-called taxonomic binning problem. Although machine learning approaches have been shown to be competitive in terms of speed and accuracy, it remains to be seen whether they can keep up to the continually increasing number of referenced genomes and species. A bottleneck for neural network models with a large number of outcome classes is the last softmax layer where probabilities are computed for all classes. The hierarchical softmax (HSM) is an approximation to the softmax that scales well with the number of classes, bringing significant improvements in training and test times. It comes at the cost of slight drop in accuracy, which can be alleviated if the hierarchy is well chosen. We adapt the hierarchical softmax by using the taxonomic tree to group classes into clusters. This new loss shows increased classification accuracy over the frequency-based hierarchical softmax, at no loss in speed.

Résumé

Nous présentons ici une nouvelle fonction objectif structurée pour l'étiquetage de fragments d'ADN de séquençage en classes taxonomiques – problème dit du *taxonomic binning*. Alors que les approches d'apprentissage statistique ont su se montrer compétitives en termes de vitesse et de précision, il reste à voir encore si elle peuvent le rester avec un nombre de génomes et d'espèces séquencés sans cesse grandissant. Un facteur limitant pour les modèles de réseaux de neurones avec un grand nombre de classes est le calcul des probabilités pour chaque classe dans la dernière couche softmax. Le softmax hiérarchique est une approximation du softmax classique qui s'adapte bien à un grand nombre de classes et qui améliore considérablement les temps d'entraînement et de prédiction. Il s'accompagne d'une perte en précision, qui peut être limitée en choisissant bien la hiérarchie. Nous adaptons le softmax hiérarchique en utilisant l'arbre taxonomique pour regrouper les classes en clusters. Cette nouvelle fonction objectif augmente la précision en classification par rapport au softmax hiérarchique basé sur les fréquences des classes, sans en impacter la vitesse.

4.1 Introduction

In this chapter we develop a method to learn taxonomic classifiers with a very large number of possible taxa. Although this method could be suited to any classification problem in which a hierarchy structures the outcome classes, we showcase its utility in the field of *metagenomics*, by identifying which microbial species are present in a DNA sequencing experiment.

With its drastic improvement in cost and efficiency, DNA sequencing has become a prime tool to study microbial communities. Modern DNA sequencing machines – so-called Next Generation Sequencing (NGS) – typically output up to the billions of short DNA sequences – *reads* – in random order. *Taxonomic binning* [Mande et al., 2012] is the task of assigning each of these reads to a taxonomic unit – a bacterial species for example. To do so reads are compared to an annotated reference genome database. As sequencing experiments are becoming commonplace, this reference database is getting increasingly large, with new genomes assembled from sequencing experiments being uploaded everyday [Szarvas et al., 2020]. Furthermore our knowledge of the tree of life is constantly expanding with the discovery of new species [Parks et al., 2018], which complicates the task of taxonomic classifiers.

Standard approaches to taxonomic binning can be roughly divided in three categories. The first are alignment-based methods, that use string-matching algorithms to align reads to a selection of reference genomes with sequence alignment software such as BWA-MEM [Li, 2013] or Bowtie [Langmead et al., 2009a]. The second category of methods, pseudo-alignment [Wood and Salzberg, 2014]; [Ounit et al., 2015]; [Kim et al., 2016]: [Bray et al., 2016], build databases of long discriminative k-mers [1] and store the taxa they are seen in. The k-mers from a read are looked up in this database and if matches are found, the corresponding taxa are returned. Finally the third class of methods formulate taxonomic binning as a machine learning classification problem [Wang et al., 2007]; [McHardy et al., 2007]. In particular, large-scale linear methods [Vervier et al., 2016] and neural network classifiers [Menegaux and Vert, 2019]; [Liang et al., 2020] are competitive on modern datasets. Although these methods are fast, both their training and testing times are affected by having a large number of output classes.

The bottleneck with neural networks with large output spaces – well known in neural language modeling where the outcome classes are the whole word vocabulary – is the normalization of the class scores into probabilities, typically done with a softmax function. The computational complexity of this normalization step scales linearly with the number of output classes, and can make up a significant portion of the training and testing times [Jozefowicz et al., 2016]. Several approaches have tackled this problem, either by (i) approximating the normalization factor by importance sampling [Bengio and Senécal, 2003]; [Mikolov et al., 2013b]; [Ji et al., 2016], (ii) replacing the softmax by self-normalized losses [Gutmann and Hyvärinen, 2010]; [Mnih and Teh, 2012] and (iii) replacing the softmax by an approximation such as the hierarchical softmax (HSM, [Goodman, 2001], [Morin and Bengio, 2005]). Our approach is an adaptation of the latter to taxonomic classification.

The hierarchical softmax groups outcome labels into clusters and decomposes classification in steps: first predicting the cluster and then choosing the label from that cluster. Introducing more levels in the hierarchy can improve the computational complexity up to the order of $\mathcal{O}\left(T/\log_2(T)\right)$ compared to the regular softmax, where T is the number of labels or taxa. However the classification accuracy does slightly drop off, probably due to error propagation

[1] A k-mer is a contiguous subsequence of length k

down the hierarchy levels. Designing the class hierarchy is primordial to mitigate this issue [Mnih and Hinton, 2009]. On one hand some approaches group classes according to their frequency, placing more common classes higher up in the tree, which has the double effect of speeding up computation and increasing the accuracy for these classes [Mikolov et al., 2011a]; [Mikolov et al., 2013a]; [Grave et al., 2017]. Another intuitive approach is to group the classes by similarity [Mnih and Hinton, 2009], so that the clusters themselves are informative. This has shown better classification accuracy than simply grouping them by frequency [Zweig and Makarychev, 2013]; [Chen et al., 2016].

Objectively defining class similarity can be challenging for some domains, however in taxonomic binning there is a natural hierarchy between classes: the phylogenetic tree. We leverage this structure in a new loss, Phylo-HS, incorporating the phylogenetic tree in the hierarchical softmax. The contributions of this chapter are as follows:

- We give a simple algorithm to adapt the taxonomic tree to the hierarchical softmax.
- We show in experiments on large-scale datasets that the resulting loss outperforms the frequency-based HSM in classification accuracy, while maintaining the same or even slightly better classification and training speed.

After briefly reviewing related work in section 2, we will present the hierarchical softmax model and our method in detail in section 3. We present the experimental setup and results in section 4.

4.2 Related Work

We first review what initially motivated this work, which is the approximation of the softmax layer and the hierarchical softmax in particular. We then give an overview of other methods using a taxonomic tree for hierarchical classification.

4.2.1 Approximating the softmax

In neural network terminology, the softmax layer is a linear classifier: the composition of the softmax function with a linear operator $h \to w \cdot h$. Effectively it gives scores for each class t then normalizes them to sum to 1. In all that follows, we will write h as the d-dimensional input to the softmax layer, and $w \in \mathbb{R}^{T \times d}$ as its parameters.

$$p_{softmax}(t \mid h) = \frac{e^{w_t \cdot h}}{\sum_{j=1}^{T} e^{w_j \cdot h}} \tag{4.1}$$

The denominator in (4.1) is also called the partition function $Z(h; w)$. Computing this term has a complexity proportional to the number of output classes and the input dimension $\mathcal{O}(dT)$. We now give the main lines of the many approaches that have been proposed to reduce this, most of which are reviewed in [Chen et al., 2016].

Reducing the effective input dimension d Some methods have aimed at reducing the complexity of the individual dot products $w_j \cdot h$. The differentiated softmax [Chen et al., 2016] or the adaptive softmax [Grave et al., 2017] both assign a lower capacity $w \in \mathbb{R}^{d'}, d' < d$ to infrequent labels. The input h is then projected to a lower dimension before computing the dot products. If T' labels are assigned this smaller representation, the complexity of computing Z and its gradients is reduced to $\mathcal{O}\left(d'T' + d(T - T')\right)$.

Approximating the partition function Another popular approach is to approximate the denominator by importance sampling [Bengio and Senécal, 2003];[Mikolov et al., 2013b];[Ji et al., 2016]: the sum in Z is taken over a random subset of the samples instead. Although these methods do speed up training, at test time the true softmax must still be evaluated.

Self-normalized classifiers [Devlin et al., 2014] maximize the unnormalized softmax likelihood with an added penalty for the normalization term $(\log Z)^2$. At test-time only the unnormalized scores are computed, forgoing the normalization completely. Other methods such as Noise Contrastive Estimation (NCE) [Gutmann and Hyvärinen, 2010]; [Mnih and Teh, 2012] change the loss function altogether and choose a binary classifier that distinguishes between the real training distribution and a noise distribution.

Hierarchical Softmax The hierarchical softmax model (HSM) is an approximation to the softmax function that groups labels into clusters. When given an input t, the model predicts its cluster c with a first classifier $p(c \,|\, h)$, then predicts the label t with a sub-model $p(t \,|\, c,\, h)$. Since each label t belongs to a unique cluster $c(t)$, the likelihood of t is the product

$$p(t \,|\, h) = p(c(t) \,|\, h)\; p(t \,|\, c(t), h)$$

If the clusters are well balanced, this reduces the complexity of the softmax from $\mathcal{O}\left(dT\right)$ to $\mathcal{O}(d\sqrt{T})$. Introducing more levels in the hierarchy further lowers the complexity, up to $\mathcal{O}\left(d \log T\right)$ for a balanced binary tree [Morin and Bengio, 2005]. The assignment of labels to clusters is usually based either on their similarities [Mnih and Hinton, 2009] or on their frequencies [Mikolov et al., 2011b], with the former exhibiting higher accuracy [Chen et al., 2016]. We review the HSM in more detail in section 4.3.1.

Although much faster than the standard softmax, the HSM comes at a price in accuracy [Mikolov et al., 2011b]; [Zweig and Makarychev, 2013]. A solution proposed in [Le et al., 2011]; [Grave et al., 2017] strikes a balance between the hierarchical speedup and the softmax accuracy by introducing a short-list, treating frequent labels as single-label clusters.

4.2.2 Leveraging label taxonomy for classification

In machine learning We focus here on classification problems in which the outcome classes are organized in a known tree hierarchy. This could in theory give extra information for classification which is not taken into account by traditional "flat" classifiers. In all generality the classes can be in the internal nodes in the tree, which calls for strategies to make predictions at different levels. [Silla and Freitas, 2011] give an extensive review of hierarchical classification across a variety of fields. More recently, [Wehrmann et al., 2018] designed a neural network architecture specially suited for multi-label classification. In the cases where the classes of interest are only in the

leaf nodes of the tree, [Wu et al., 2017] give a hierarchical loss that penalizes errors differently depending on how far in the tree the prediction is from the true label. For instance predicting "dog" instead of "cat" is more harshly penalized than predicting "skyscraper".

Taxonomic binning in metagenomics In the metagenomics problem we are interested in, outcome classes are microbial taxa (such as bacterial species) which are organized in a taxonomic tree. Ideally predictions at finer levels such as species or genus are more desirable but making predictions at coarser levels such as phylum or family can also be of value. Indeed, some short DNA fragments can be shared by genomes of multiple species, in which case a prediction at a higher level can represent this ambiguity. Taxonomic binning software such as Kraken [Wood and Salzberg, 2014] and Centrifuge [Kim et al., 2016] both use the taxonomic tree to resolve multiple possible matches by returning the Least Common Ancestor (LCA) of all the matching taxa. Another way to use the taxonomic tree at test time is to sum estimated species-level probabilities to make predictions at the genus level [Busia et al., 2018]. This simple approach can be used by any binner outputting probabilities. In the most sophisticated approach yet, [Rojas-carulla et al., 2018] introduce a cascading softmax with separate softmax layers classifying each taxonomic rank r. Each softmax layer takes as input the hidden representation h and the output of the previous softmax layer. The loss function is a weighted average of the output softmaxes. This however does not reduce the computational complexity and on the contrary scales with the total number of nodes in the taxonomic tree rather than just the leaves.

4.3 Methods

4.3.1 Hierarchical softmax

We suppose established a tree hierarchy $\mathcal{T}$ over T labels. Reusing the notations in [Silla and Freitas, 2011], for a given vertex $t \in \mathcal{T}$, we define $\uparrow(t) \in \mathcal{T}$ as its parent, and $\downarrow(t)$ as the set of its children. We further denote $r(t) \in \mathbb{N}$ as its depth – its distance to the root t_0. With these notations, applying the function $\uparrow r(t)$ times to any vertex t returns to the root: $\uparrow^{r(t)}(t) = t_0$.

For each internal node t, the hierarchical softmax model learns a classifier $\hat{f}_t$ that distinguishes between its children given an input $h \in \mathbb{R}^d$. Each of these classifiers is a softmax model with a $|\downarrow(t)| \times d$ matrix of weights w:

$$\forall i \in \downarrow(t), \quad \hat{f}_t(h)_i = \frac{\exp h \cdot w_i}{\sum_{j \in \downarrow(t)} \exp h \cdot w_j} \tag{4.2}$$

The computational complexity of each $\hat{f}_t$ therefore depends on its number of children $\mathcal{O}\left(d\,|\downarrow(t)|\right)$.

Under this model, the likelihood of any given node is the product of the conditional probabilities of all the nodes in its lineage:

$$p(t \mid h) = \prod_{i=1}^{r(t)} p(\uparrow^{i-1}(t) \mid \uparrow^{i}(t), h) \tag{4.3}$$

$$= \prod_{i=1}^{r(t)} \left[\hat{f}_{\uparrow^i(t)}(h)\right]_{\uparrow^{i-1}(t)} \tag{4.4}$$

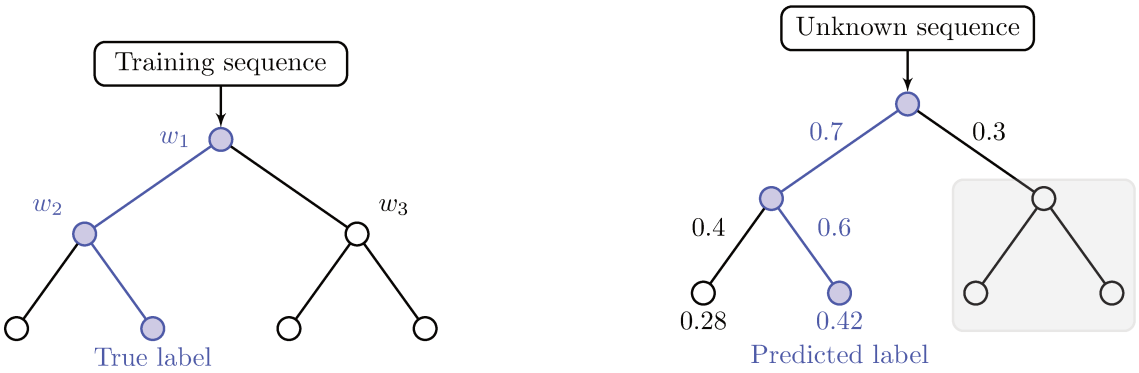


Figure 4.1 – Toy example of a binary classification tree. On the left, we highlight the traversal of the tree for a training example with a known label. Only the weights w_1 and w_2 are used and updated during the gradient descent. w_3 is not used at all. On the right we show predicting the most probable label for a test sequence. Leaf probabibilities are computed in a depth-first traversal. We show the state here after the probabilities on the left side are computed. Since the best label has a greater probability $0.42 > 0.3$ than the right child of the root, the deeper children on the right do not have to be computed (shaded area)

These parameters are learned jointly by maximizing the log-likelihood of the taxa in the training set (usually the leaves) through stochastic gradient descent (SGD).

Computational complexity If the average number of children per ancestor node is C, the number of operations in the computation of the likelihood and gradient per sample t is $\mathcal{O}(d \times r(t) \times C)$. The quantity $r(t) \times C$ is minimized for binary trees, where it drops to $\log_2(T)$ on average. One can further reduce the average $r(t)$ by placing more frequent labels higher up in the tree. [Mikolov et al., 2013a] push this to the limit by Huffman coding the labels, which ensures an optimal average path length w.r.t. the empirical distribution. This tree, which we refer to as the Huffman tree, is also the one used in the hierarchical softmax loss of `fastText` [Joulin et al., 2016].

A detail perhaps worth mentioning is that binarizing the tree lowers the number of parameters for the HSM. In the non-binary version, a vector of d-dimensional weights is learned for each node except the root, so the total number of parameters for the HSM is $(|\mathcal{T}| - 1) \times d$. For the binary tree, this is reduced to $(T - 1) \times d$ as each local classifier $\hat{f}_t$ is a binary logistic model. Compared to the worst case scenario (a ternary tree), this corresponds to a reduction of 33%.

It is important to note that, at test-time, computing the probability for every class will still be an $\mathcal{O}(T)$ operation. However if one is interested only in the top few most probable classes, some parts of the tree can be bypassed, as shown in figure 4.1. This is a factor to consider when designing the hierarchy $\mathcal{T}$. For instance in the Huffman tree probabilities are computed for the most frequent labels first. Building a hierarchy based on similarity can also boost this by funneling the probability structure, rather than dispersing the probabilities higher up in the tree.

4.3.2 Tree construction

Extracting from the reference tree of life The classification tree for Phylo-HS is constructed at training time from the list of taxonomic IDs of training genomes, and a reference tree of life $\mathcal{T}_{ref}$, ideally containing all species of interest. For each taxon t in the training set, we query the

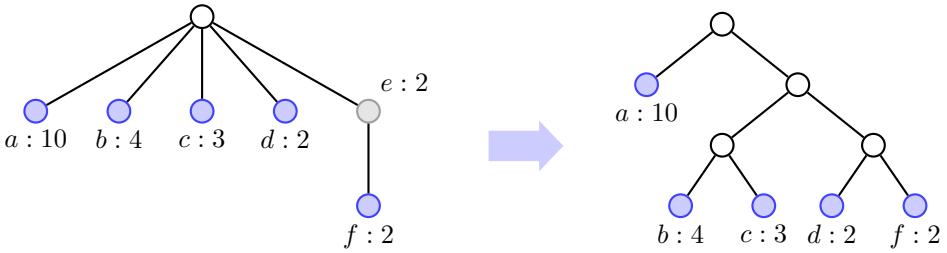


Figure 4.2 – Example of binarization by frequency. The leaf taxons $a; b; c; d; f$ are highlighted in blue, with the number of occurrences in the training dataset. For Phylo-HS, these frequencies are proportional to the total amount of bp belonging to genomes with taxon $a; \ldots; f$. On the *left*, they are laid out as originally. Taxon e has only one child and therefore is removed from the tree. On the *right*, the taxons as appearing in the final binary tree. Both children of each node should have a balanced total frequency.

reference tree to get its full *lineage*, its path to the root $l(t) = \left[t, \uparrow(t), \ldots, \uparrow^{r(t)}(t)\right]$ The new tree $\mathcal{T}_{hs}$ is the minimal tree covering all training taxa and their ancestors. We further simplify $\mathcal{T}_{hs}$ by removing all nodes that have only one child.

Binarizing the taxonomic tree In order to attain the maximal exponential speedup, we transform the resulting tree into a *full* binary tree, in which each node has either 0 or exactly 2 children. As shown in figure 4.2, leaf nodes are left untouched and nodes having only one child are removed. For nodes that have more than 2 children, we split the siblings into two separate groups according to their frequencies (total length of their representative genomes in training set). The C children are ordered by descending frequency $[1, \ldots, C]$ and then split into two groups $[1, \ldots, K]$ and $[K+1, \ldots, C]$, where K is the first child such that $|1| + \ldots + |K| > |K+1| + \ldots |C|$. This ensures that the total frequencies of each group are balanced, and that more frequent classes are on average higher up in the tree, hence accessed more rapidly. As the taxonomic hierarchy is conserved, the resulting tree can be seen as a hybrid between the Huffman coding tree and the taxonomic tree.

4.4 Results

4.4.1 Learning the input representation

We test Phylo-HS on the taxonomic binning problem, where taxa must be assigned to short ($\sim 50 - 400$bp-long) DNA reads. A DNA read of length L is a sequence $\mathbf{x} = x_1 \ldots x_L \in \mathcal{A}^L$, where $\mathcal{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ is the alphabet of nucleotides. To transform a DNA read into an input representation $h \in \mathbb{R}^d$, we use `fastDNA` [Menegaux and Vert, 2019]. More specifically, we segment a read into overlapping k-mers – contiguous subsequences of length k. A d-dimensional embedding is learned for each of the N k-mers, stored in an $N \times d$ embeddings matrix $M = (M_u)_{u \in \mathcal{A}^k}$. The representation $h = \Phi^M(\mathbf{x})$ of a read $\mathbf{x}$ is then simply the average of the embeddings of its constituent k-mers:

$$\forall \mathbf{x} \in \mathcal{A}^L\,, \quad \Phi^M(\mathbf{x}) = \sum_{i=1}^{L-k+1} M_{\mathbf{x}_{[i,i+k-1]}}\,. \tag{4.5}$$

The embeddings matrix will be jointly trained by SGD with the output layer, either the full softmax layer or the HSM presented above. Training reads are extracted on the fly from genomes in the training set, with the addition of some random noise in the form of substitutions at a uniform rate.

4.4.2 Data

We test our method on 6 datasets of increasing complexity, composed by downloading genomes from the NCBI RefSeq database [NCBI Resource Coordinators, 2016]. The first 3 are *small*, *medium* and *large* from [Vervier et al., 2016], with respectively 51, 193 and 771 species represented in 356, 1564 and 2768 complete genomes. We will also use the validation set for *large* – composed of 193 genomes not present in the training set but whose species are – to evaluate the classification performance of the models. While the *large* dataset gave a comprehensive picture of the known bacterial and archeal genomes at the time of its publication in 2015, reference databases have since expanded.

As a more realistic dataset we use the 10,857 genomes of 3,640 species from the DeepMicrobes preprint [Liang et al., 2019]. It was obtained by filtering out similar species of the RefSeq database, by comparing the similarity of one of their representative genomes. Our largest dataset is *RefSeq CG* from the benchmark [Ye et al., 2019]. It contains all the complete microbial genomes available in RefSeq as of 2018, a total of 23000 genomes belonging to 13000 taxa. As a large portion of those taxa are viruses having relatively small genomes, we simplify this dataset by attributing the same domain-level taxonomic id (label) to all viruses. The resulting dataset contains the same number of genomes with 5500 taxa.

As a reference taxonomic tree, we use the NCBI taxonomy downloaded on 30/09/2020.

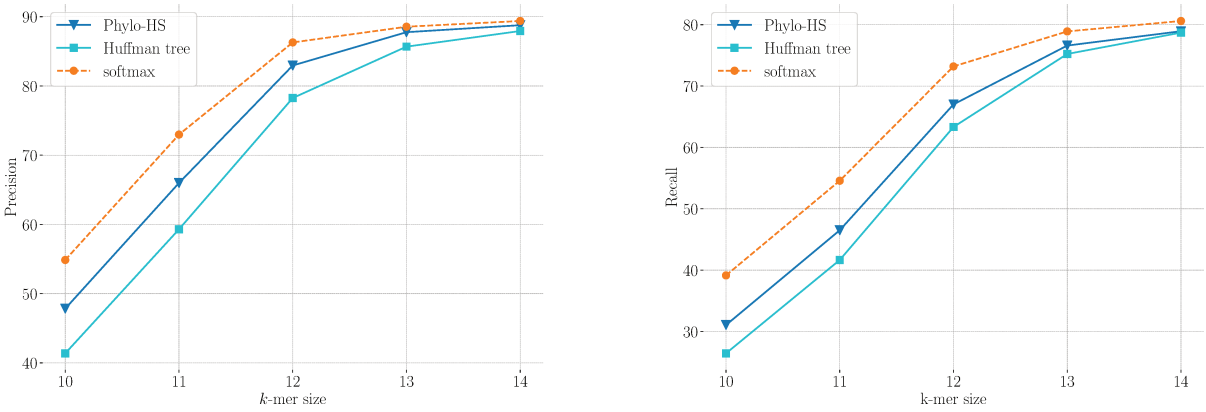
4.4.3 Classification accuracy

The measures reported here are of species-level recall and precision. For a given species t, if n_{true} is the number of validation sequences of species t, n_{pred} is the number of sequences classified as t by the model, and finally n_{tp} is the number of true positives: sequences correctly classified as t. Recall (or sensitivity) for a species is the proportion n_{tp}/n_{true}. Average species-level recall is obtained by averaging this value over all the species truly present in the validation set. Precision is the proportion n_{tp}/n_{pred}. Average species-level precision is obtained by averaging this value over all the predicted species.

In figure 4.3, `fastDNA` models were trained for the same number of epochs on the *large* dataset then evaluated on the *large* validation set. One epoch corresponds to a number of reads sufficient to cover on average once all the positions in the training genomes. Phylo-HS does better both in terms of precision and recall than the Huffman coding alternative, although the gap does narrow as the model capacity gets higher (larger k). Similar results have been reported for language models [Mnih and Hinton, 2009]; [Chen et al., 2016]. This does not however fully compensate the performance dropoff of the hierarchical softmax compared to the full softmax.

4.4.4 Classification speed

All speed tests were performed on a Intel Xeon CPU E5-2440 0 - 2.40GHz with 12 cores and 128GB of RAM. Models were trained using 12 threads, and tested with a single thread. There



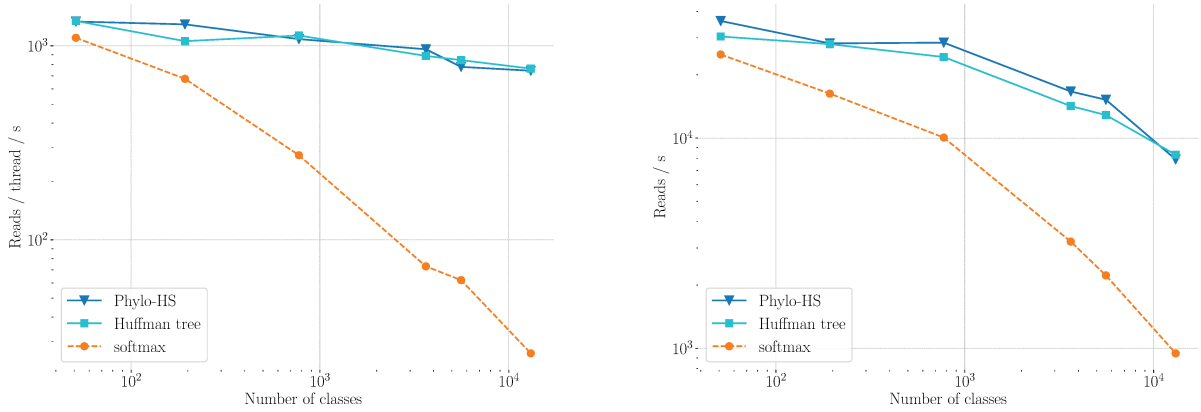
a Average species-level precision

b Average species-level recall

Figure 4.3 – Comparison of the effect of the loss function on the classification performance a `fastDNA` model, on the *large* dataset. Results are shown as a function of the k-mer size k. The models, all of embedding dimension $d = 50$, were trained for 100 epochs with a learning rate of 0.1 and a mutation noise rate of 4%.

was some variability in our speed measurements, so we report the median time over 10 runs. Models all have the same hyperparameters $k = 14$ and $d = 50$ and were trained for 100 epochs. We do not include fixed costs such as model loading time to concentrate solely the asymptotic per-read speed. These fixed costs are the same for the studied models.

As confirmed in figure 4.4, training the full softmax for the larger label sets becomes prohibitively slow, while the hierarchical softmax models are much less affected. Both versions of the hierarchical softmax are equally fast in both training and testing. While there is perhaps a slight edge for Phylo-HS at testing time, this could be attributed to variance in the speed measurements or data-specific quirks. As expected, the speed improvement with respect to the full softmax does diminish between test and training, but it remains a substantial 10× speedup on the largest dataset.



a Reads per second per thread processed at training

b Reads per second processed at testing

Figure 4.4 – Comparison of the effect of the loss function on the number of reads per second processed by a `fastDNA` model. Results are shown as a function of the number of output classes. All models have k-mer size $k = 14$ and embedding dimension $d = 50$. Input reads are 200-bp long.

4.5 Discussion and future work

We have shown with these preliminary experiments that modelling the hierarchical softmax on the taxonomic tree can bring gains in classification accuracy at no cost in computational complexity. However it must be noted that the frequency-based trees such as the Huffman coding one do still have the advantage of being nomenclature-agnostic. Indeed they do not depend on labels being recognized as official taxonomic IDs, which makes them potentially better suited for characterizing genomes of unknown species. One could still apply Phylo-HS in that case by clustering genomes based on simple measures of similarity such as Average Nucleotide Identity (ANI) for instance. In fact many tools have been developed to infer phylogenetic trees from raw genomes [Ankenbrand and Keller, 2016]; [Na et al., 2018]; [Szarvas et al., 2020], and in particular for metagenome assembled genomes (MAGs) [Wu, 2018]. It could be a promising approach to use these tools on the training genomes to build the taxonomic classification tree.

An added benefit of Phylo-HS is its inherent ability to make predictions at any taxonomic rank. This could either be specified by the user or determined by the model at prediction time. One way to do this would be to give thresholds p_r for each rank, and stop the prediction at a node of rank r if it does not have any children of probability greater than p_r.

The code for constructing the Phylo-HS tree and training the `fastDNA` model are freely available online.

Chapter 5

Conclusion

In the past fifteen years, the technological leap in next generation sequencing has enabled large-scale characterization of ecosystems through their genomic information. Efficient bioinformatics techniques are needed to process the large quantity of data output by these experiments. As machine learning methods have shown great success in many applications, steadily replacing or complementing more traditional methods, this thesis aims to help this transition in the domain of high-throughput sequencing and of metagenomics in particular. We present new supervised learning tools to classify DNA sequencing reads into taxonomic units, and propose scalable solutions as the number of sequenced genomes and known species continues to expand rapidly.

In chapter 2, we develop a simple short read classifier `fastDNA`. DNA fragments are represented by the vector of their k-mer counts. These representations are then given to a neural network with one hidden layer of small dimension d – the embedding layer. We implement a general training procedure which extracts reads on the fly from the full reference genomes, injecting noise in the form of random mutations. We show in our experiments that for the right embedding dimension d and k-mer size, the resulting model is comparable to state of the art taxonomic binners in terms of speed and accuracy. k and d are jointly constrained by memory considerations, with larger models yielding the best classification accuracy.

In chapter 3, we extend the previous chapter by allowing for longer k-mers. Since the number of k-mers becomes prohibitively large for large k, we introduce a strategy to group similar k-mers together and give them the same representation. This strategy is based on the de Bruijn graph: k-mers belonging to the same contigs – linear substretches of the graph – are merged together. We show that `fastDNA` models trained with longer k exhibit higher precision at the cost of lower recall. Unfortunately building the de Bruijn graph for a very large number of disparate genomes is not currently feasible on a standard computer, requiring too much memory. `Brume` may therefore be more adequate on smaller datasets, or to distinguish similar genomes at a finer resolution.

Finally in chapter 4, we attempt to tackle two challenges at the same time: (i) reducing the training and testing times of neural network based taxonomic binners for problems with a large number of output classes and (ii) leveraging the inherent hierarchical structure in these classes to improve classifier accuracy. Our proposed approach incorporates the taxonomic tree in a hierarchical linear classifier, learning local sub-models at each node of the taxonomic tree. These models learn to distinguish between their respective children. We show the resulting classifier outperforms frequency-based hierarchical models in terms of accuracy while maintaining their speed improvement.

We now conclude this thesis with an outlook into future research directions related to our contributions.

Beyond k-mers: other DNA tokenizations Most statistical learning approaches for long contiguous sequences first require breaking up the input sequence into sub-units, or tokens. Although human language has natural breakpoints with spaces and punctuation, biological sequences do not. The most intuitive DNA tokens – fixed-length contiguous subsequences, or k-mers – have proven their worth in many applications. However they can be redundant for large k. Some methods use LSH-based hashing techniques to give the same representation to similar k-mers (w.r.t the Hamming distance). Although these methods have been proposed in machine learning models in [Georgiou et al., 2020] and [Shi and Chen, 2019], they have not been compared thoroughly with regular k-mers.

An inconvenience in k-mers or their derivatives is choosing an appropriate k. Too short and they are not expressive enough, too long and they become too numberous and less robust to noise. Another altogether new method would be to use a DNA variant of word segmentation techniques such as WordPiece [Wu et al., 2016] which would break any sequence into non-overlapping k-mers of different lengths. This k-mer vocabulary will have been chosen to maximize the likelihood of a standard language model on input genomes under a fixed cardinality constraint. In future work, we aim to implement these different tokenizations and evaluate their effects on the performance of machine learning models.

Multi-rank predictions and whole sample approaches We hinted in chapter 4 that Phylo-HS could potentially be used as a rank-flexible approach, making predictions at higher ranks when it is not sure about the lower ones. This could be beneficial to reattribute these ambiguous reads after seeing the composition of the whole metagenomic sample. In this thesis we have mostly focused on classifying reads one by one independently of one another, but more global approaches might be better suited depending on the application. An exciting research direction that has already been explored in [Queyrel et al., 2020] and [Georgiou et al., 2020] is to treat the whole sample all together, either for abundance estimations or clinical diagnosis. In this type of Multiple Instance Learning problem, particular care will be given to learning intermediary representations of reads such as the ones computed with `fastDNA`, with more sophisticated attention mechanisms.

Standardized pipelines for machine learning in taxonomic binning A significant amount of time in this thesis was spent on preparing training data and comparing new models to the existing state of the art. We believe this is currently done over and over again by researchers in independent fashion, who would greatly benefit from more accessible pipelines. We found it was not easy to compare to existing approaches, as each are trained and evaluated on their own custom datasets, often using different performance metrics. Fortunately the publication of benchmarks such as [Sczyrba et al, 2018] and [Ye et al., 2019] have greatly helped this issue. We hope the metagenomics community will continue to expand on these efforts and make model evaluation both more thorough and more straightforward.

As modern machine learning models require a large amount of data to be properly tuned, designing a proper data pipeline is a challenge in itself. As existing read simulating tools are not specifically adapted to machine learning, many papers implement their own methods. Indeed current read simulators generate large text files with a given amount of reads, which becomes impractical. A very useful practical project would therefore be to open-source an on-the-fly read simulator that could be easily plugged in to machine learning pipelines.

Ideally with such methods available researchers and practitioners could spend more time on

the developmental phases of algorithm design, and less on the field-specific data processing and experimenting, hopefully helping to make machine learning in bioinformatics more appealing as a whole to incoming students and researchers.

Bibliography

B. Alipanahi, A. Delong, M. T. Weirauch and B. J. Frey. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nature Biotechnology*, 33(8):831–838, 2015. ISSN 1087-0156. doi: 10.1038/nbt.3300. URL http://dx.doi.org/10.1038/nbt.3300. 20

S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990. ISSN 00222836. doi: 10.1016/S0022-2836(05)80360-2. 10

S. L. Amarasinghe, S. Su, X. Dong, L. Zappia, M. E. Ritchie and Q. Gouil. Opportunities and challenges in long-read sequencing data analysis. *Genome Biology*, 21(1):30, 2020. ISSN 1474-760X. doi: 10.1186/s13059-020-1935-5. URL https://doi.org/10.1186/s13059-020-1935-5. 6

F. E. Angly, D. Willner, F. Rohwer, P. Hugenholtz and G. W. Tyson. Grinder: a versatile amplicon and shotgun sequence simulator. *Nucleic Acids Res.*, 40:e94, 2012. 8, 35, 52

M. J. Ankenbrand and A. Keller. bcgTree: automatized phylogenetic tree building from bacterial core genomes. *Genome*, 59(10):783–791, may 2016. ISSN 0831-2796. doi: 10.1139/gen-2015-0175. URL https://doi.org/10.1139/gen-2015-0175. 71

E. Asgari and M. R. Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PLoS ONE*, 10(11):1–15, 2015. ISSN 19326203. doi: 10.1371/journal.pone.0141287. 19

N. A. Baeshen, M. N. Baeshen, A. Sheikh, R. S. Bora, M. M. M. Ahmed, H. A. I. Ramadan, K. S. Saini and E. M. Redwan. Cell factories for insulin production. *Microbial Cell Factories*, 2014. URL http://www.microbialcellfactories.com/content/13/1/141. 3

A. Baevski and M. Auli. Adaptive input representations for neural language modeling. *arXiv*, pages 1–13, 2018. ISSN 23318422. 26

D. Bahdanau, K. H. Cho and Y. Bengio. Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015. 20

S. Balzer, K. Malde, A. Lanzén, A. Sharma and I. Jonassen. Characteristics of 454 pyrosequencing data–enabling realistic simulation with flowsim. *Bioinformatics*, 26:i420–i425, 2010. 35, 52

Y. Bengio and J.-S. Senécal. Quick Training of Probabilistic Neural Nets by Importance Sampling. *AISTATS*, 2003. 26, 63, 65

D. Blakely, E. Collins, R. Singh, A. Norton, J. Lanchantin and Y. Qi. FastSK : fast sequence analysis with gapped string kernels. *Bioinformatics*, 36(26), 2020. doi: 10.1093/bioinformatics/btaa817. 18

P. Bojanowski, E. Grave, A. Joulin and T. Mikolov. Enriching Word Vectors with Subword Information. *arXiv*, 2016. ISSN 10450823. doi: 1511.09249v1. URL http://arxiv.org/abs/1607.04606. 19

P. Bojanowski, E. Grave, A. Joulin and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. ISSN 2307-387X. 32

N. L. Bray, H. Pimentel, P. Melsted and L. Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016. ISSN 15461696. doi: 10.1038/nbt.3519. 22, 24, 26, 51, 63

K. Brinda, M. Sykulski and G. Kucherov. Spaced seeds improve k -mer-based metagenomic classification. *Bioinformatics*, 31(July):3584–3592, 2015. doi: 10.1093/bioinformatics/btv419. 10

M. Burrows and D. Wheeler. A Block-sorting Lossless Data Compression Algorithm. *SRC Research Report*, 1994. ISSN 12106313. 9

A. Busia, G. E. Dahl, C. Fannjiang, D. H. Alexander, E. Dorfman, R. Poplin, C. Y. McLean, P.-C. Chang and M. DePristo. A deep learning approach to pattern recognition for short DNA sequences. *bioRxiv preprint*, 2018. 20, 66

K. Břinda, M. Baym and G. Kucherov. Simplitigs as an efficient and scalable representation of de Bruijn graphs. *bioRxiv preprint*, pages 1–30, 2020. 26

S. Canzar and S. L. Salzberg. Short Read Mapping: An Algorithmic Tour. *Proc IEEE Inst Electr Electron Eng.*, 176:436–458, 2017. doi: 10.1109/JPROC.2015.2455551.Short. 10

E. P. Caragata and T. Walker. Using bacteria to treat diseases. *Expert Opin Biol Ther*, 12(6):701–712, Jun 2012. 3

K. Chen and L. Pachter. Bioinformatics for whole-genome shotgun sequencing of microbial communities. *PLoS Comput Biol*, 1(2):106–112, Jul 2005. 4

W. Chen, D. Grangier and M. Auli. Strategies for training large vocabulary neural language models. *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Long Papers*, 4:1975–1985, 2016. doi: 10.18653/v1/p16-1186. 64, 65, 69

A. Conneau. Very Deep Convolutional Networks for Text Classification. *arXiv*, 2016. 20

A. Corvelo, W. E. Clarke, N. Robine and M. C. Zody. TaxMaps - Ultra-comprehensive and highly accurate taxonomic classification of short-read data in reasonable time. *bioRxiv*, 2017. ISSN 1088-9051. doi: 10.1101/134023. 11

J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz and J. Makhoul. Fast and robust neural network joint models for statistical machine translation. *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, 1:1370–1380, 2014. doi: 10.3115/v1/p14-1129. 65

J. Devlin, M.-w. Chang, L. Kenton and K. Toutanouva. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv*, 2017. 22

J. A. Eisen. Environmental shotgun sequencing: its potential and challenges for studying the hidden world of microbes. *PLoS Biol*, 5(3):e82, Mar 2007. 6

J. L. Elman. Finding structure in time. *Cognitive Science*, 1990. ISSN 03640213. doi: 10.1016/0364-0213(90)90002-E. 20

G. Eraslan, Ž. Avsec, J. Gagneur and F. J. Theis. Deep learning: new computational modelling techniques for genomics. *Nature Reviews Genetics*, 2019. doi: 10.1038/s41576-019-0122-6. 21

M. Escalona, S. Rocha and D. Posada. Europe PMC Funders Group Europe PMC Funders Author Manuscripts A comparison of tools for the simulation of genomic next- generation sequencing data. *Nature Reviews Genetics*, 17(8):459–469, 2017. doi: 10.1038/nrg.2016.57.A. 8

Y. Fan and O. Pedersen. Gut microbiota in human metabolic health and disease. *Nature Reviews Microbiology*, 19(1):55–71, 2021. ISSN 1740-1534. doi: 10.1038/s41579-020-0433-9. URL https://doi.org/10.1038/s41579-020-0433-9. 3

P. Ferragina and G. Manzini. Opportunistic data structures with applications. *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 390–398, 2000. doi: 10.1109/SFCS.2000.892127. 9

A. Fritz, P. Hofmann, S. Majda, E. Dahms, J. Dröge, J. Fiedler, T. R. Lesker, P. Belmann, M. Z. Demaere et al. CAMISIM: Simulating metagenomes and microbial communities. *bioRxiv*, pages 1–12, 2018. doi: 10.1101/300970. 8

A. Georgiou, V. Fortuin, A. Mustafa and G. Rätsch. Meta 2 : Memory-efficient taxonomic classification and abundance estimation for metagenomics with deep learning. *arXiv*, 2020. 19, 22, 24, 74

M. Ghandi, D. Lee, M. Mohammad-noori and M. A. Beer. Enhanced Regulatory Sequence Prediction Using Gapped k-mer Features. *PLOS Computational Biology*, 10(7), 2014. doi: 10.1371/journal.pcbi.1003711. 10, 18

A. Goglio, M. Tucci, B. Rizzi, A. Colombo, P. Cristiani and A. Schievano. Microbial recycling cells (mrcs): A new platform of microbial electrochemical technologies based on biocompatible materials, aimed at cycling carbon and nutrients in agro-food systems. *Science of The Total Environment*, 649:1349 – 1361, 2019. ISSN 0048-9697. doi: https://doi.org/10.1016/j.scitotenv.2018.08.324. URL http://www.sciencedirect.com/science/article/pii/S0048969718332959. 3

J. Goodman. Classes for fast maximum entropy training. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 1:561–564, 2001. ISSN 15206149. doi: 10.1109/icassp.2001.940893. 27, 63

S. Goodwin, J. D. McPherson and W. R. McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351, 2016. ISSN 1471-0064. doi: 10.1038/nrg.2016.49. URL https://doi.org/10.1038/nrg.2016.49. 6, 7

E. Grave, A. Joulin and M. Cissé. Efficient softmax approximation for GPUs. *arXiv*, 2017. 26, 64, 65

I. Gregor, J. Dröge, M. Schirmer, C. Quince and A. C. Mchardy. PhyloPythiaS + : a self-training method for the rapid reconstruction of low-ranking taxonomic bins from metagenomes. *PeerJ*, pages 1–21, 2016. doi: 10.7717/peerj.1603. 18

N. H. W. Group, J. Peterson, S. Garges, M. Giovanni, P. McInnes, L. Wang, J. A. Schloss, V. Bonazzi, J. E. McEwen et al. The NIH human microbiome project. *Genome research*, 19: 2317–2323, Dec. 2009. 31

S. Gupta, P. Gupta and V. Pruthi. *Microbial Production of Antibiotics Using Metabolic Engineering*, pages 205–213. Springer Singapore, Singapore, 2020. ISBN 978-981-15-2604-6. doi: 10.1007/978-981-15-2604-6_13. URL https://doi.org/10.1007/978-981-15-2604-6_13. 3

M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Journal of Machine Learning Research*, 2010. 63, 65

S. Hochreiter and J. J. Urgen Schmidhuber. Long short term memory. Neural computation. *MEMORY Neural Computation*, 1997. 20

G. Holley. Bifrost – Highly parallel construction and indexing of colored and compacted de Bruijn graphs. *Genome Biology*, 2020. 24, 57

L. A. Hug, B. J. Baker, K. Anantharaman, C. T. Brown, A. J. Probst, C. J. Castelle, C. N. Butterfield, A. W. Hernsdorf, Y. Amano et al. A new view of the tree of life. *Nature Microbiology*, 1(5):1–6, 2016. ISSN 20585276. doi: 10.1038/nmicrobiol.2016.48. 4, 8

P. Hugenholtz, B. M. Goebel and N. R. Pace. Impact of culture-independent studies on the emerging phylogenetic view of bacterial diversity. *J Bacteriol*, 180(18):4765–4774, Sep 1998. 3

D. H. Huson, A. F. Auch, J. Qi and S. C. Schuster. MEGAN analysis of metagenomic data. *Genome Res.*, 17:377–386, 2007. 31

C. Huttenhower, D. Gevers, R. Knight, S. Abubucker, J. H. Badger, A. T. Chinwalla, H. H. Creasy, A. M. Earl, M. G. FitzGerald et al. Structure, function and diversity of the healthy human microbiome. *Nature*, 486(7402):207–214, Jun 2012. 3

R. M. Idury and M. S. Waterman. A New Algorithm for DNA Sequence Assembly. *Journal of Computational Biology*, 1995. ISSN 15578666. doi: 10.1089/cmb.1995.2.291. 24

H. Jegou, M. Douze and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. 35

S. Ji, S. V. Vishwanathan, N. Satish, M. J. Anderson and P. Dubey. Blackout: Speeding up recurrent neural network language models with very large vocabularies. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pages 1–14, 2016. 16, 26, 63, 65

Y. Ji, Z. Zhou, H. Liu and R. V. Davuluri. DNABERT : pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome. *bioRxiv preprint*, 2020. 22

D. T. Jones and S. M. Kandathil. High precision in protein contact prediction using fully convolutional neural networks and minimal sequence features. *Bioinformatics*, 34(April): 3308–3315, 2018. doi: 10.1093/bioinformatics/bty341. 20

A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou and T. Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016. 24, 32, 34, 35, 67

A. Joulin, E. Grave, P. Bojanowski and T. Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics, April 2017. 19, 34

J. Jovel, J. Patterson, W. Wang, N. Hotte, S. O'Keefe, T. Mitchel, T. Perry, D. Kao, A. L. Mason et al. Characterization of the gut microbiome using 16s or shotgun metagenomics. *Frontiers in Microbiology*, 7:459, 2016. ISSN 1664-302X. doi: 10.3389/fmicb.2016.00459. URL https://www.frontiersin.org/article/10.3389/fmicb.2016.00459. 8

R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer and Y. Wu. Exploring the Limits of Language Modeling. *arXiv*, 2016. URL http://arxiv.org/abs/1602.02410. 20, 63

D. R. Kelley, J. Snoek and J. L. Rinn. Basset : learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Research*, pages 990–999, 2016. doi: 10.1101/gr.200535.115.Freely. 20

D. Kim, L. Song, F. P. Breitwieser and S. L. Salzberg. Centrifuge : rapid and sensitive classification of metagenomic sequences. *Genome Resesarch*, pages 1–9, 2016. doi: 10.1101/gr.210641.116. Freely. 11, 63, 66

E. V. Koonin. *The logic of chance: the nature and origin of biological evolution*. FT press, 2011. 6

J. Korbel, A. Abyzov, X. Mu, N. Carriero, P. Cayting, Z. Zhang, Z. Snyder and M. Gerstein. PEMer: a computational framework with simulation-based error models for inferring genomic structural variants from massive paired-end sequencing data. *Genome Biol.*, 10(2):R23, 2009. 35, 52

D. J. Lane, B. Pace, G. J. Olsen, D. A. Stahl, M. L. Sogin and N. R. Pace. Rapid determination of 16S ribosomal RNA sequences for phylogenetic analyses. *Proc Natl Acad Sci U S A*, 82(20): 6955–6959, Oct 1985. 8

B. Langmead, C. Trapnell, M. Pop and S. L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3), 2009a. ISSN 14747596. doi: 10.1186/gb-2009-10-3-r25. 10, 46, 63

B. Langmead, C. Trapnell, M. Pop and S. L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009b. 31

N. LaPierre, M. Alser, E. Eskin, D. Koslicki and M. Serghei. Metalign: Efficient alignment-based metagenomic profiling via containment min hash. *bioRxiv*, 2020. 10

H.-S. Le, I. Oparin, A. Allauzen, J.-L. Gauvain and F. Yvon. Structured output layer neural network language model. *ICASSP*, 2011. ISSN 2618-8627. 65

C. Leslie, E. Eskin and W. Noble. The spectrum kernel: a string kernel for SVM protein classification. In R. B. Altman, A. K. Dunker, L. Hunter, K. Lauerdale and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing 2002*, pages 564–575, Singapore, 2002. World Scientific. 18, 32

C. Leslie, E. Eskin, J. Weston and W. Noble. Mismatch String Kernels for SVM Protein Classification. In S. Becker, S. Thrun and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press, 2003. 10, 18, 33

H. Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. Technical Report 1303.3997, arXiv, 2013. 10, 35, 46, 52, 63

H. Li. Minimap2 : pairwise alignment for nucleotide sequences. *arXiv*, 2018. 10, 46

H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25:1754–1760, 2009. 31

Q. Liang, P. W. Bible, Y. Liu, B. Zou and L. Wei. DeepMicrobes : taxonomic classification for metagenomics with deep learning. *bioRxiv preprint*, 2019. 20, 22, 46, 69

Q. Liang, P. W. Bible, Y. Liu, B. Zou and L. Wei. DeepMicrobes: taxonomic classification for metagenomics with deep learning. *NAR Genomics and Bioinformatics*, 2(1):1–13, 2020. doi: 10.1093/nargab/lqaa009. 63

J. Lu, F. P. Breitwieser, P. Thielen and S. L. Salzberg. Bracken: Estimating species abundance in metagenomics data. *PeerJ Computer Science*, 2017(1), 2017. ISSN 23765992. doi: 10.7717/peerj-cs.104. 22

Y. Luo, Y. W. Yu, J. Zeng, B. Berger and J. Peng. Metagenomic binning through low density hashing. *bioRxiv*, 2017. URL `http://biorxiv.org/content/early/2017/05/02/133116.abstract`. 15, 18, 40

S. S. Mande, M. H. Mohammed and T. S. Ghosh. Classification of metagenomic sequences: methods and challenges. *Briefings Bioinf*, 13:669–681, 2012. 31, 63

A. C. McHardy, H. G. Martín, A. Tsirigos, P. Hugenholtz and I. Rigoutsos. Accurate phylogenetic classification of variable-length DNA fragments. *Nat. Methods*, 4(1):63–72, 2007. 31, 32, 63

L. McInnes, J. Healy and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv*, 2018. ISSN 23318422. 15

R. Menegaux and J.-P. Vert. Continuous Embeddings of DNA Sequencing Reads and Application to Metagenomics. *J Comput Biol*, 26(6):509–518, 06 2019. 19, 46, 47, 48, 52, 63, 68

R. Menegaux and J.-P. Vert. Embedding the de bruijn graph, and applications to metagenomics. *bioRxiv*, mar 2020. doi: 10.1101/2020.03.06.980979.

P. Menzel, K. L. Ng and A. Krogh. Fast and sensitive taxonomic classification for metagenomics with Kaiju. *Nature Communications*, 7, 2016. ISSN 20411723. doi: 10.1038/ncomms11257. 11

T. Mikolov, A. Deoras, D. Povey, L. Burget and J. Černocký. Strategies for training large scale neural network language models. *2011 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2011, Proceedings*, pages 196–201, 2011a. doi: 10.1109/ASRU.2011.6163930. 64

T. Mikolov, S. Kombrink, L. Burget, J. Černocký and S. Khudanpur. Extensions of recurrent neural network language model. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2011b. ISBN 9781457705397. doi: 10.1109/ICASSP.2011.5947611. 65

T. Mikolov, K. Chen, G. Corrado and J. Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv*, pages 1–12, 2013a. ISSN 15324435. doi: 10.1162/153244303322533223. URL http://arxiv.org/abs/1301.3781. 18, 32, 64, 67

T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean. Distributed representations ofwords and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 2013b. 26, 27, 28, 63, 65

A. Milanese, D. R. Mende, L. Paoli, G. Salazar, H.-j. Ruscheweyh, M. Cuenca, P. Hingamp, R. Alves, P. I. Costea et al. Microbial abundance, activity and population genomic profiling with mOTUs2. *Nature Communications*, 2019. ISSN 2041-1723. doi: 10.1038/s41467-019-08844-4. URL http://dx.doi.org/10.1038/s41467-019-08844-4. 11

M. Mirdita, M. Steinegger, F. Breitwieser and L. K. E. Fast and sensitive taxonomic assignment to metagenomic contigs. *bioRxiv preprint*, 2020. 11

K. Miyazaki and N. Tomariguchi. Occurrence of randomly recombined functional 16S rRNA genes in Thermus thermophilus suggests genetic interoperability and promiscuity of bacterial 16S rRNAs. *Nature research, Scientific Reports*, 2019. ISSN 2045-2322. doi: 10.1038/s41598-019-47807-z. URL http://dx.doi.org/10.1038/s41598-019-47807-z. 8

A. Mnih and G. Hinton. A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*, 2009. ISBN 9781605609492. 64, 65, 69

A. Mnih and Y. W. Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 2012. ISBN 9781450312851. 63, 65

F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *AISTATS 2005 - Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, 2005. ISBN 097273581X. 27, 63, 65

S.-I. Na, Y. O. Kim, S.-H. Yoon, S.-m. Ha, I. Baek and J. Chun. UBCG: Up-to-date bacterial core gene set and pipeline for phylogenomic tree reconstruction. *Journal of Microbiology*, 56 (4):280–285, 2018. ISSN 1976-3794. doi: 10.1007/s12275-018-8014-6. URL https://doi.org/10.1007/s12275-018-8014-6. 71

NCBI Resource Coordinators. Database resources of the National Center for Biotechnology Information. *Nucleic acids research*, 44(D1):D7–D19, jan 2016. ISSN 1362-4962. doi: 10.1093/nar/gkv1290. URL https://pubmed.ncbi.nlm.nih.gov/26615191https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4702911/. 9, 69

R. Ounit, S. Wanamaker, T. J. Close and S. Lonardi. CLARK : fast and accurate classification of metagenomic and genomic sequences using discriminative k -mers. *BMC Genomics*, 2015. doi: 10.1186/s12864-015-1419-2. 46, 63

X. Pan, P. Rijnbeek, J. Yan and H. B. Shen. Prediction of RNA-protein sequence and structure binding preferences using deep convolutional and recurrent neural networks. *BMC Genomics*, 2018. ISSN 14712164. doi: 10.1186/s12864-018-4889-1. 20

D. H. Parks, N. J. MacDonald and R. G. Beiko. Classifying short genomic fragments from novel lineages using composition and homology. *BMC Bioinf.*, 12:328, 2011. 31, 32

D. H. Parks, M. Chuvochina, D. W. Waite, C. Rinke, A. Skarshewski, P.-A. Chaumeil and P. Hugenholtz. A standardized bacterial taxonomy based on genome phylogeny substantially revises the tree of life. *Nature Biotechnology*, 36(10):996–1004, 2018. ISSN 1546-1696. doi: 10.1038/nbt.4229. URL https://doi.org/10.1038/nbt.4229. 4, 63

K. R. Patil, L. Roune and A. C. McHardy. The PhyloPythiaS web server for taxonomic assignment of metagenome sequences. *PLoS One*, 7:e38581, 2012. 31, 32

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee and L. Zettlemoyer. Deep contextualized word representations. *arXiv*, 2018. 20

J. Qin, R. Li, J. Raes, M. Arumugam, K. S. Burgdorf, C. Manichanh, T. Nielsen, N. Pons, F. Levenez et al. A human gut microbial gene catalogue established by metagenomic sequencing. *Nature*, 464(7285):59–65, 2010. ISSN 1476-4687. doi: 10.1038/nature08821. URL https://doi.org/10.1038/nature08821. 4

D. Quang and X. Xie. DanQ: A hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic Acids Research*, 2016. ISSN 13624962. doi: 10.1093/nar/gkw226. 20

M. Queyrel, E. Prifti and J.-d. Zucker. Towards end-to-end disease prediction from raw metagenomic data. *bioRxiv preprint*, 2020. 22, 74

A. Radford, K. Narasimhan, T. Salimans and I. Sutskever. Improving Language Understanding by Generative Pre-Training. *arXiv*, 2018. 22

C. S. Riesenfeld, P. D. Schloss and J. Handelsman. Metagenomics: genomic analysis of microbial communities. *Annu. Rev. Genet.*, 38:525–552, 2004. 31

A. Rives, J. Meier, T. Sercu, S. Goyal, Z. Lin, D. Guo, M. Ott, C. L. Zitnick, J. Ma et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv preprint*, 2020. 22

M. Rojas-carulla, I. Tolstikhin, G. Luque, N. Youngblut, L. Ruth and B. Sch¨olkopf. GeNet : Deep Representations for Metagenomics. *arXiv*, 2018. 20, 66

L. Schaeffer, H. Pimentel, N. Bray, P. All and L. Pachter. Pseudoalignment for metagenomic read assignment ´. *arXiv*, pages 1–13, 2015. 11, 26

K. Scott and C. Murano. Microbial fuel cells utilising carbohydrates. *Journal of Chemical Technology and Biotechnology*, 82:92 – 100, 01 2007. doi: 10.1002/jctb.1641. 3

A. Sczyrba et al. Critical Assessment of Metagenome Interpretation – a benchmark of computational metagenomics software. *Nature Methods*, 14(11):1063–1071, 2018. doi: 10.1038/nmeth.4458.Critical. 12, 74

A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. R. Nelson et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020. ISSN 1476-4687. doi: 10.1038/s41586-019-1923-7. URL https://doi.org/10.1038/s41586-019-1923-7. 22

A. Shcherbina. FASTQSim: platform-independent data characterization and in silico read generation for NGS datasets. *BMC Research Notes*, 7(1):533, 2014. ISSN 1756-0500. doi: 10.1186/1756-0500-7-533. URL https://doi.org/10.1186/1756-0500-7-533. 8

L. Shi and B. Chen. A Vector Representation of DNA Sequences Using Locality Sensitive Hashing. *bioRxiv preprint*, 2019. 19, 24, 74

C. N. Silla and A. A. Freitas. A survey of hierarchical classification across different application domains, 2011. ISSN 13845810. 65, 66

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014. ISSN 15337928. 16

S. Subrata, W. Zigeng and R. Sanguthevar. A novel algorithm to accurately classify metagenomic sequences. *bioRxiv preprint*, 2020. 11

J. Szarvas, J. Ahrenfeldt, J. L. B. Cisneros, M. C. F. Thomsen, F. M. Aarestrup and O. Lund. Large scale automated phylogenomic analysis of bacterial isolates and the Evergreen Online platform. *Communications Biology*, 3(1):137, 2020. ISSN 2399-3642. doi: 10.1038/s42003-020-0869-5. URL https://doi.org/10.1038/s42003-020-0869-5. 63, 71

The 1000 Genomes Project Consortium, A. Auton, L. D. Brooks, R. M. Durbin, E. P. Garrison, H. M. Kang, J. O. Korbel, J. L. Marchini, S. McCarthy et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, oct 2015. ISSN 1476-4687. doi: 10.1038/nature15393.

URL https://pubmed.ncbi.nlm.nih.gov/26432245https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4750478/. 6

D. T. Truong, E. A. Franzosa, T. L. Tickle, M. Scholz, G. Weingart, E. Pasolli, A. Tett, C. Huttenhower and N. Segata. MetaPhlAn2 for enhanced metagenomic taxonomic profiling. *Nature Methods*, 12(10):902–903, 2015. ISSN 1548-7105. doi: 10.1038/nmeth.3589. URL https://doi.org/10.1038/nmeth.3589. 11

E. J. Vandamme. Production of vitamins, coenzymes and related biochemicals by biotechnological processes. *J Chem Technol Biotechnol*, 53(4):313–327, 1992. 3

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin. Attention Is All You Need. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017. 20, 22

K. Vervier, P. Mahé, M. Tournoud, J.-B. Veyrieras and J.-P. Vert. Large-scale machine learning for metagenomics sequence classification. *Bioinformatics*, 32:1023–1032, 2016. 18, 23, 31, 32, 34, 35, 39, 46, 48, 52, 63, 69

Q. Wang, G. M. Garrity, J. M. Tiedje and J. R. Cole. Naive bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Appl. Environ. Microbiol.*, 73:5261–5267, 2007. 18, 31, 32, 63

J. Wehrmann, R. Cerri and R. C. Barros. Hierarchical multi-label classification networks. In *35th International Conference on Machine Learning, ICML 2018*, 2018. ISBN 9781510867963. 65

D. E. Wood and S. L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biol.*, 15(3):R46, 2014. 11, 26, 34, 46, 63, 66

D. E. Wood, J. Lu and B. Langmead. Improved metagenomic analysis with Kraken 2. *bioRxiv preprint*, 2019. 11, 46

C. Wu, M. Tygert and Y. LeCun. Hierarchical loss for classification, 2017. ISSN 23318422. 66

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv*, pages 1–23, 2016. URL http://arxiv.org/abs/1609.08144. 22, 74

Y. W. Wu. ezTree: An automated pipeline for identifying phylogenetic marker genes and inferring evolutionary relationships among uncultivated prokaryotic draft genomes. *BMC Genomics*, 19 (Suppl 1), 2018. ISSN 14712164. doi: 10.1186/s12864-017-4327-9. 71

J. Xu. Distance-based protein folding powered by deep learning. *PNAS*, 116(34), 2019. doi: 10.1073/pnas.1821309116. 20

S. H. Ye, K. J. Siddle, D. J. Park and P. C. Sabeti. Primer Benchmarking Metagenomics Tools for Taxonomic Classification. *Cell*, 178(4):779–794, 2019. ISSN 0092-8674. doi: 10.1016/j.cell.2019.07.010. URL https://doi.org/10.1016/j.cell.2019.07.010. 8, 12, 69, 74

M. Zaheer, S. Kottur and S. Ravanbhakhsh. Deep Sets. *arXiv*, 2017. 22

H. Zhou, A. Shrikumar and A. Kundaje. Benchmarking Reverse-Complement Strategies for Deep Learning Models in Genomics. *bioRxiv*, page 2020.11.04.368803, 2020. URL https://doi.org/10.1101/2020.11.04.368803. 18

J. Zhou and O. G. Troyanskaya. Predicting effects of noncoding variants with deep learning– based sequence model. *Nature Methods*, 12(10):931–934, 2016. doi: 10.1038/nmeth.3547.Predicting. 20

C. Zimmer. How microbes defend and define us, 2010. URL https://www.nytimes.com/2010/07/13/science/13micro.html. 3

G. Zweig and K. Makarychev. Speed regularization and optimality in word classing. *arXiv*, pages 8237–8241, 2013. 64, 65

RÉSUMÉ

Le coût du séquençage de l'ADN a été divisé par 100 000 en seulement 15 ans. Grâce à cette révolution technologique, des volumes de données toujours plus grands émergent, posant de nouvelles problématiques informatiques. Comment analyser et stocker les séquences d'ADN de manière efficiente ? La *métagénomique*, qui cherche à caractériser et identifier les microbes – bactéries, virus – à partir de leur ADN, a largement bénéficié de cette avancée. Une expérience de séquençage produit des milliards de petits fragments d'ADN (reads), mélangés aléatoirement. Une étape cruciale en bioinformatique est d'identifier le génome d'origine de chaque fragment, un problème dit de *taxonomic binning*. Les méthodes classiques, basées sur l'alignement des séquences à des génomes de référence, devenues trop lentes avec l'augmentation du nombre de génomes, ont été remplacées par le pseudo-alignement. Celui-ci cherche des sous-séquences du read dans une base préexistante.
L'apprentissage statistique offre également des résultats prometteurs pour la classification des séquences biologiques. Dans cette thèse, nous approfondirons ces méthodes pour le taxonomic binning. Nous présenterons d'abord **fastDNA**, un algorithme qui apprend des représentations continues pour tous les k-mers (courtes sous-séquences de longueur k, ou "mots" de l'ADN). Une représentation vectorielle du read est obtenue en combinant celles de ses k-mers, et un classifieur linéaire en prédit la classe. Ensuite, nous introduirons **Brume**, une extension de fastDNA qui regroupe les k-mers via le graphe de de Bruijn, augmentant le nombre de k-mers effectifs sans surcoût mémoire. Enfin, nous présenterons **Phylo-HS**, une nouvelle fonction d'apprentissage statistique basée sur l'arbre phylogénétique.

MOTS CLÉS

métagénomique, apprentissage statistique, séquençage haut débit, représentations vectorielles

ABSTRACT

The cost of DNA sequencing has been divided by 100,000 in the past 15 years. Brought along by this technological revolution, ever larger volumes of data are coming in from diverse fields and problems, raising new computational challenges. How can we efficiently store and analyze DNA sequences? A modern DNA sequencing experiment outputs billions of short DNA fragments (reads), in random order. A crucial step in the bioinformatics analysis pipeline is to match those fragments to their parent genomes, a problem called taxonomic binning. Up until a few years ago alignment-based strategies were the norm, which were largely based on string-matching algorithms. However these have become too slow for the ever-growing amount of available sequenced genomes. More recently so-called pseudo-alignment strategies have become standard. These hold databases of large sub-strings and look for matches in the query sequences.
Machine learning methods have shown promising success in classifying biological sequences and in this thesis we will investigate these methods for taxonomic binning. Firstly, we present an algorithm, fastDNA, that embeds sequences in a continuous vector space by first splitting them into short k-mers (substrings of length k) and learning an embedding for each k-mer. The embedding is then run through a linear classifier. In the second part of this thesis we will present Brume, an extension to fastDNA that allow for longer k-mers, using the de Bruijn graph. Finally we will introduce Phylo-HS, a structured loss for neural network-based taxonomic classification.

KEYWORDS

metagenomics, statistical learning, DNA sequencing, vector embeddings