DISSERTATION

MODERN CONSIDERATIONS FOR THE USE OF NAIVE BAYES IN THE SUPERVISED CLASSIFICATION OF GENETIC SEQUENCE DATA

Submitted by

Steven M. Lakin

Department of Microbiology, Immunology and Pathology

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2021

Doctoral Committee:

Advisor: Zaid Abdo

Sanjay Rajopadhye
Mark Stenglein
Jane Stewart

ABSTRACT

MODERN CONSIDERATIONS FOR THE USE OF NAIVE BAYES IN THE SUPERVISED CLASSIFICATION OF GENETIC SEQUENCE DATA

Genetic sequence classification is the task of assigning a known genetic label to an unknown genetic sequence. Often, this is the first step in genetic sequence analysis and is critical to understanding data produced by molecular techniques like high throughput sequencing. Here, we explore an algorithm called naive Bayes that was historically successful in classifying 16S ribosomal gene sequences for microbiome analysis. We extend the naive Bayes classifier to perform the task of general sequence classification by leveraging advancements in computational parallelism and the statistical distributions that underlie naive Bayes. In Chapter 2, we show that our implementation of naive Bayes, called WarpNL, performs within a margin of error of modern classifiers like Kraken2 and local alignment. We discuss five crucial aspects of genetic sequence classification and show how these areas affect classifier performance: the query data, the reference sequence database, the feature encoding method, the classification algorithm, and access to computational resources. In Chapter 3, we cover the critical computational advancements introduced in WarpNL that make it efficient in a modern computing framework. This includes efficient feature encoding, introduction of a log-odds ratio for comparison of naive Bayes posterior estimates, description of schema for parallel and distributed naive Bayes architectures, and use of machine learning classifiers to perform outgroup sequence classification. Finally in Chapter 4, we explore a variant of the Dirichlet multinomial distribution that underlies the naive Bayes likelihood, called the beta-Liouville multinomial. We show that the beta-Liouville multinomial can be used to enhance classifier performance, and we provide mathematical proofs regarding its convergence during maximum likelihood estimation. Overall, this work explores the naive Bayes algorithm in a modern context and shows that it is competitive for genetic sequence classification.

ACKNOWLEDGEMENTS

I am extraordinarily thankful to the individuals who have supported me in my educational endeavors, including my immediate and extended family, my significant other, and all of my mentors, adivsers, teachers, and labmates along the way. I thank my committee and major professor for their thoughtful instruction and dedication to my academic career. Finally, I thank the contributions of the various funding agencies who have supported my course of study, the educational programs that contributed to my success, and the many others who are too numerous to name here.

DEDICATION

I dedicate this work to those who have supported me along this path and to the pursuit of knowledge for its own sake.

TABLE OF CONTENTS

Chapter 1

Introduction

Genetic sequence classification is fascinating in its deceptive simplicity. On the surface, this problem is a simple comparison of strings: how many edits does it take to get from sequence A to sequence B. This is an algorithm taught in introductory computer science courses that has well-defined answers; but for genetic sequences, behind this simple facade is a rich problem that drives at deep structural patterns in evolutionary theory. When we ask the question of "how many edits" it takes to get from A to B, we are asking a question of distance: what is the distance between A and B?

Distance is a well-studied geometric topic, and calculating a distance between two points is simple when the geometric space is a familiar one, such as in Euclidean geometries. However, in non-Euclidean spaces, the answer to the question of distance is not always clear or easy to find. Perhaps in the dense city blocks of Manhattan, the shortest path between two points is not a straight line but a path following the roads. Likewise, if the question is of distance travelled on foot between two points on Earth, certainly we must follow the natural topology when calculating our answer. And what for genetic sequences? We understand how to analyze complex geometries in high dimensional spaces, but what do genetic tensors look like, and how can we calculate distances between them?

The key to understanding genetic geometries is in defining an appropriate measure of evolutionary distance. The problem with using simple edit distance for genetic sequences is that not all edits have the same cost (Figure 1.1). Changes to a single nucleotide in the genetic code can have drastic changes on protein tertiary structure; and since proteins are the workhorses of life, detrimental changes to their function have important implications for organismal viability. Thus, some changes to nucleic acid sequences cost more than others from an evolutionary perspective. This phenomenon is well-studied and has already been incorporated into early bioinformatics applications via the work of Margaret Dayhoff in 1978, who introduced the concept of point accepted

mutation matrices (PAM) that described the various costs associated with nucleotide and protein edits [1].

In fact, this discussion of evolutionary cost will sound familiar to those who study phylogenetics, since these concepts form the foundation of the molecular clock hypothesis [2]. There has been extensive work in phylogenetics on evolutionary models that offer answers to our question of genetic distance. However, these models either require a great degree of prior knowledge regarding the biological system under study or a dataset capturing variation in time and space (and the other aspects of Hardy-Weinberg equilibrium) from which to infer the model parameters [3]. What if our two sequences come from different evolutionary systems or nothing is known of the selective pressures to which they are subjected? Are they even evolutionarily comparable, and if so, in what way should we calculate a distance between them?

Thus, the apparently simple task of comparing genetic sequences is in reality a Herculean feat: given no prior information about the sequences other than what can be inferred from the sequences themselves, calculate a distance that accurately captures the evolutionary constraints required to get from sequence A to sequence B. This is the task that we ask of genetic sequence classifiers. These classifiers, which have their mathematical roots in well-studied geometries like Euclidean space, have nearly zero chance of unraveling this problem without substantial help from outside information. This is why it is critical to find a transformation or a way of using evolutionary information that will help the algorithms map genetic tensors to a more familiar geometry (Figure 1.1). Arguably, this is exactly what the PAM and BLOSUM matrices do for alignment algorithms; alignment is perhaps still in wide use today primarily due to the information captured by these underlying cost matrices and only secondarily due to the success of the algorithmic techniques themselves.

It is clear that successful genetic sequence classifiers will achieve at least one of the following objectives: 1) the algorithm will learn to capture the evolutionary information required to calculate an accurate distance between the two sequences being compared, or 2) we will find a way to map the sequences from genetic space into more familiar geometries where these algorithms have been

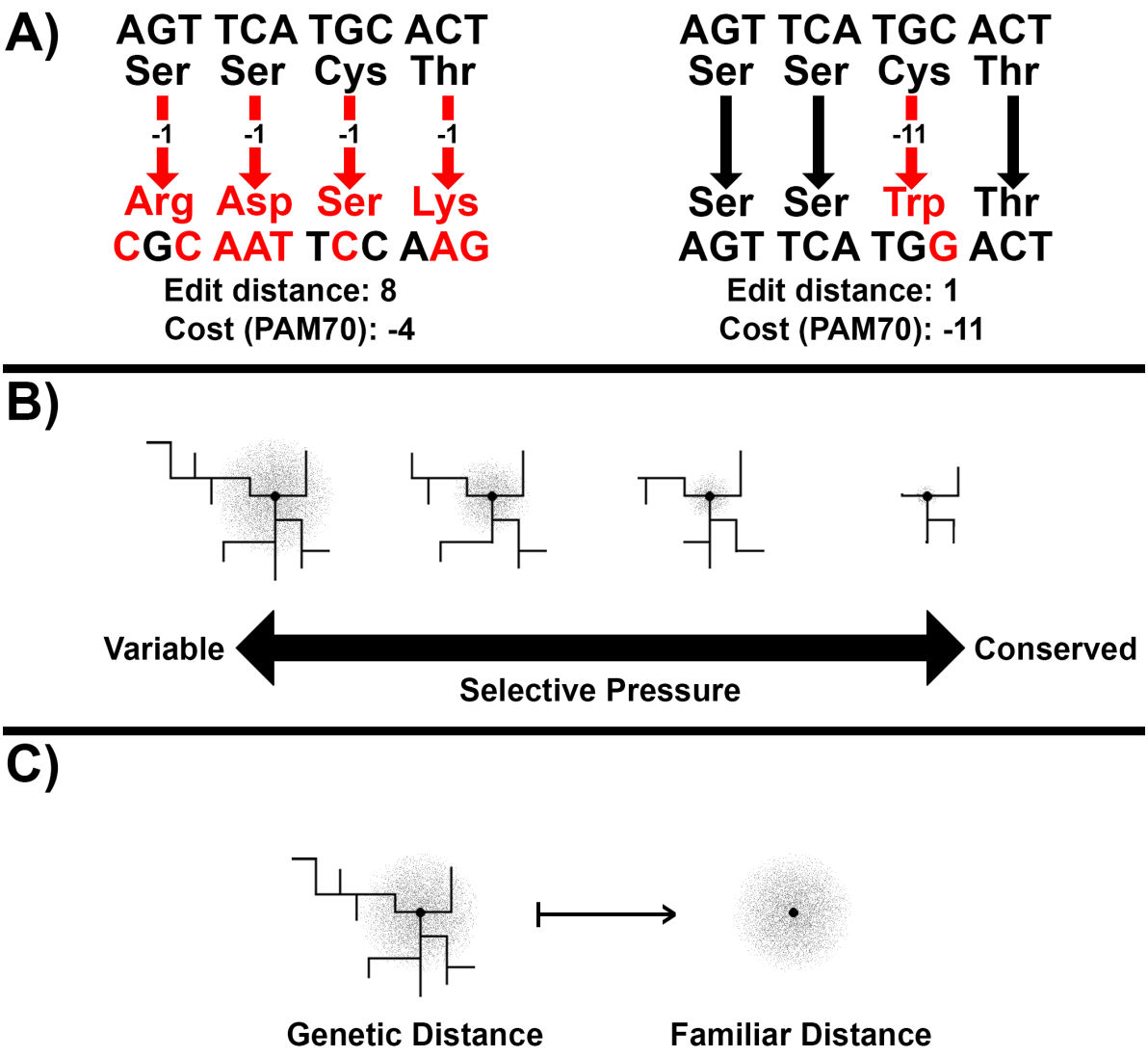

Figure 1.1: Genetic distance does not necessarily correlate with edit distance; therefore genetic geometries are unfamiliar to modern algorithms, which were developed to handle geometries with familiar distance metrics, such as Euclidean distance. A) For genetic sequences, edit distance is the number of nucleotide changes required to get from sequence A to sequence B. However, due to evolutionary systems under selective pressure, not all edits cost the same. Therefore, edit distance does not often correlate well with evolutionary distance or cost. Here, we show a hypothetical situation where an edit distance of 8 has an evolutionary cost of only -4 (left), while an edit distance of 1 has an evolutionary cost of -11 (right). These costs were produced with the point accepted mutation (PAM70) matrix, which uses multiple sequence alignments to produce an evolutionary cost metric for changes in protein sequences. B) For a given point in genetic sequence space (dot), this diagram shows a hypothetical example of traveling a fixed distance away from that point, holding evolutionary cost constant. Some paths are longer (edit distance) but cost the same as shorter paths from an evolutionary standpoint. As selective pressure increases in a system, the number of allowed mutations decreases, since evolutionary cost increases with every edit. C) If an accurate mapping can be found from genetic space to a familiar geometry, it will enable classification algorithms to perform much more accurately. The PAM and BLOSUM matrices are examples of putative mappings.

shown to perform well. In hindsight, it is perhaps this second objective on which I should have focused my studies, however I only reached this conclusion due to the substantial efforts contained in this work. Instead, I chose to focus on the first objective: to find an algorithmic approach that might be able to better capture evolutionary information and therefore better perform genetic sequence classification.

I chose to revisit the naive Bayes classifier, since it had performed successfully in constrained tasks like 16S gene classification for microbiome analysis [4]. In this more constrained setting, the input sequences were known to be 16S genes due to selective molecular amplification, and small differences between these genetic sequences separated them by sometimes great evolutionary distances. If the naive Bayes approach was capable of matching small changes in genetic sequences to their appropriate phylogenetic clade, it stood to reason that it might perform similarly well in the general sense.

Chapter 2 presents a modern implementation of the naive Bayes classifier for general sequence classification and compares it to well-known classification approaches that are in widespread use in the bioinformatics field. The thesis of Chapter 2 is to make the following important point: algorithms are only lenses through which we view genetic geometries; they cannot change the structure of the data once it is created. Algorithms are critically important to genetic sequence classification, however they are only data microscopes: various implementations can provide us with different views of the data, but if the data are not amenable to easy viewing, then the algorithms cannot force them to reveal their information. The only way to change the structure of the genetic geometry is to change the input data (the query or training data) or to change how the data are described in genetic space (feature encoding). Therefore, while algorithms serve an important role, they are not magic bullets. Equally or more important to the success of genetic sequence classification are the quality of the input data and the method in which the data are presented to the underlying math that the classifiers use.

Chapter 3 describes the critical ideas that were required to implement the naive Bayes algorithm into a general and modern framework. This chapter focuses on two key components:

computational efficiency using hardware acceleration and parallel computing, and the method by which the classifier determines if a sequence should be classified or not. Without the painstakingly-implemented optimizations used in our implementation of naive Bayes, it would not have been fast or accurate enough to handle the size and complexity of modern genetic sequence datasets. In the years to come, I expect to see most classifiers take advantage of the same resources we leveraged in the implementation of modern naive Bayes, since datasets continue to grow in size, and the classification tasks continue to become more complex. Additionally, Chapter 3 outlines one strategy for determining whether a sequence should be labelled as "unclassified." This is a question that drives at the deep evolutionary structures outlined above, and it should be an area of further study. Here, we have presented one solution in a specific domain, however more general solutions to this problem may reveal deeper patterns about genetic geometries that could improve our understanding of genetic sequences as a field.

Chapter 4 explores the underlying statistical distribution used by the naive Bayes classifier: the multinomial distribution. The way in which the naive Bayes classifier "sees" the data is through the lens of the multinomial distribution; so any errant assumptions made by this distribution would affect classifier performance. Since some of these assumptions seem dubious in the context of genetic sequence classification, we compared its performance to two other variants of the multinomial: the Dirichlet multinomial and the beta-Liouville multinomial in hopes that it would improve classification accuracy in sequence-based problems. We performed evaluation of the beta-Liouville multinomial using datasets from the field of text natural language processing, since this field has a greater availability of gold standard datasets. Ultimately, we decided that any marginal gains from the Dirichlet or beta-Liouville multinomial would incur too much of a performance cost for use in the genetic sequence classification settings, since their use would require maximum likelihood estimation over a much larger feature space.

Overall, this work presents a successful implementation of the naive Bayes algorithm into a general framework for genetic sequence classification that performs at least as well as widely used methods. It explores advantages and disadvantages of the naive Bayes approach, both for classi-

fier behavior and computational resource utilization. It assesses flaws inherent in the multinomial distribution that underlies naive Bayes and proposes alternatives that may be of future use. As a whole, the work explores one potential solution to the problem of genetic sequence classification: can an algorithm be developed that captures appropriate evolutionary information during sequence comparison? While it seems that this particular implementation of naive Bayes has not fully addressed that question, this work has certainly proved instructive, at least to its author, and it will lead to further, refined investigations in the pursuit of understanding genetic geometries.

Chapter 2

Considerations for performing supervised classification of high-throughput sequencing data: why algorithms are not magic bullets

2.1 Summary

Classifying and annotating genetic sequences is a cornerstone of bioinformatics analysis of high throughput sequencing data. This task, called genetic sequence classification, is composed of five key components: the query sequence data, the reference sequence database, feature encoding strategies, the classification algorithm, and the availability of computational resources. While much attention is given to algorithms in the literature, the other four areas of genetic sequence classification are equally, if not more important, to performing accurate analysis. In this work, we discuss each of these areas and their effect on genetic sequence classification in a manner accessible to non-experts. We present a novel naive Bayes machine learning approach, called WarpNL, and compare it to several widely used classification strategies. We show how these classification approaches behave in practice and discuss how they can be utilized in combination to improve the sensitivity and accuracy of genetic sequence classification. Our discussion throughout the work is supported by concrete examples using simulated, gold-standard, and real-world datasets. Overall, we provide a clear and thorough picture of supervised classification for high throughput sequencing data that is instructive to beginners in bioinformatics but nuanced enough to provide additional insights for experienced analysts.

2.2 Author Summary

Modern investigations into genomes and populations of organisms are often performed using rapid sequencing technologies. These sequencing platforms produce a huge amount of sequence

data that must be computationally analyzed to be properly understood. The task of identifying a sequence is called genetic sequence classification and involves a combination of computer algorithms, known genomic databases, and other mathematical methods. Much of modern scientific literature and media reporting focuses on "artificial intelligence," which is the algorithmic aspect of this process. However, we show here that the other components of performing genetic sequence classification are equally critical to its success, including: the quality of the input data, the way in which the data are presented to the computer, and the computer resources available to the analyst. We introduce a new machine learning implementation, called WarpNL, and compare its behavior on various kinds of data to other widely used approaches. Throughout the work, we discuss the basics and nuances of how genetic sequence classification is performed and how it can be leveraged to improve overall results. In particular, we focus on how different algorithms behave differently and how their advantages can be leveraged together to better effect.

2.3 Introduction

High-throughput sequencing of nucleic acids is now a ubiquitous and crucial feature of many biological fields spanning academia, regulation, private industry, and public health [5–8]. For example, sequencing efforts during the SARS-CoV-2 global pandemic enabled detection and tracking of novel viral variants in real time, leading to multi-national implementation of public health guidance and diversion of efforts into novel vaccine products [9, 10]; sequencing of microbial populations increased our understanding of the pervasive and urgent threat of antimicrobial resistance [11, 12]; and, genomic characterization of cancerous tumor variation resulted in targeted therapies and early detection programs for at-risk populations [13, 14]. Given the enormous impact of sequencing on these diverse fields of study and its ubiquitous use, it is now more critical than ever for a wide audience of researchers to understand how these data are analyzed in easily understood terms.

Perhaps the most important aspect of analyzing high throughput sequence data is the step of genetic sequence classification: identifying what sequences are actually present in the data. In

this work, we present five key components of genetic sequence classification and show how they affect the classification task: the encoding of sequences into a feature space, the use of genetic sequence databases, the behavior of classification algorithms, the effect of different kinds of query sequence data, and the importance of access to computational resources (Figure 2.1). We discuss these areas in historical and modern contexts and provide real-world examples of their impact on the classification process to make it readily available to the general audience. In presenting this work, we demonstrate how different sequence classification strategies behave in practice so that the analyst or researcher can better use these methods either individually or in combination to achieve more accurate results from high-throughput sequencing data.

First, we discuss the topic of feature encoding to show how sequences are represented in machine language and how such features behave in practice. Next, we cover the crucial nature of having a robust genetic sequence database for use as reference genomes or as a training dataset for machine learning classifiers. We also provide guidelines for the creation of robust genetic sequence databases. We then cover three major categories of classification algorithms and introduce a novel approach of our own creation, called WarpNL. The algorithms discussed include local alignment (Burrows-Wheeler Aligner), partial alignment/exact matching (Kraken2), and accelerated naive Bayes machine learning (WarpNL) [15, 16]. We demonstrate the performance and practical behavior of each of these algorithms on simulated and real datasets and use the MEGARes antimicrobial resistance sequence database as an example reference dataset [17]. Finally, we discuss practical details for algorithmic implementation with currently available computational resources and methods for improving the sensitivity of sequence detection in the face of increasing sequence variation.

Our discussion emphasizes that the choice of feature encoding method and the robustness of the genetic database greatly impact classification accuracy, which should be a focus of bioinformatics research. Our results show that different categories of genetic sequence classification algorithms behave differently on different kinds of data, highlighting the need for a combined approach to achieve the most accurate and sensitive results. We also demonstrate that classification strate-

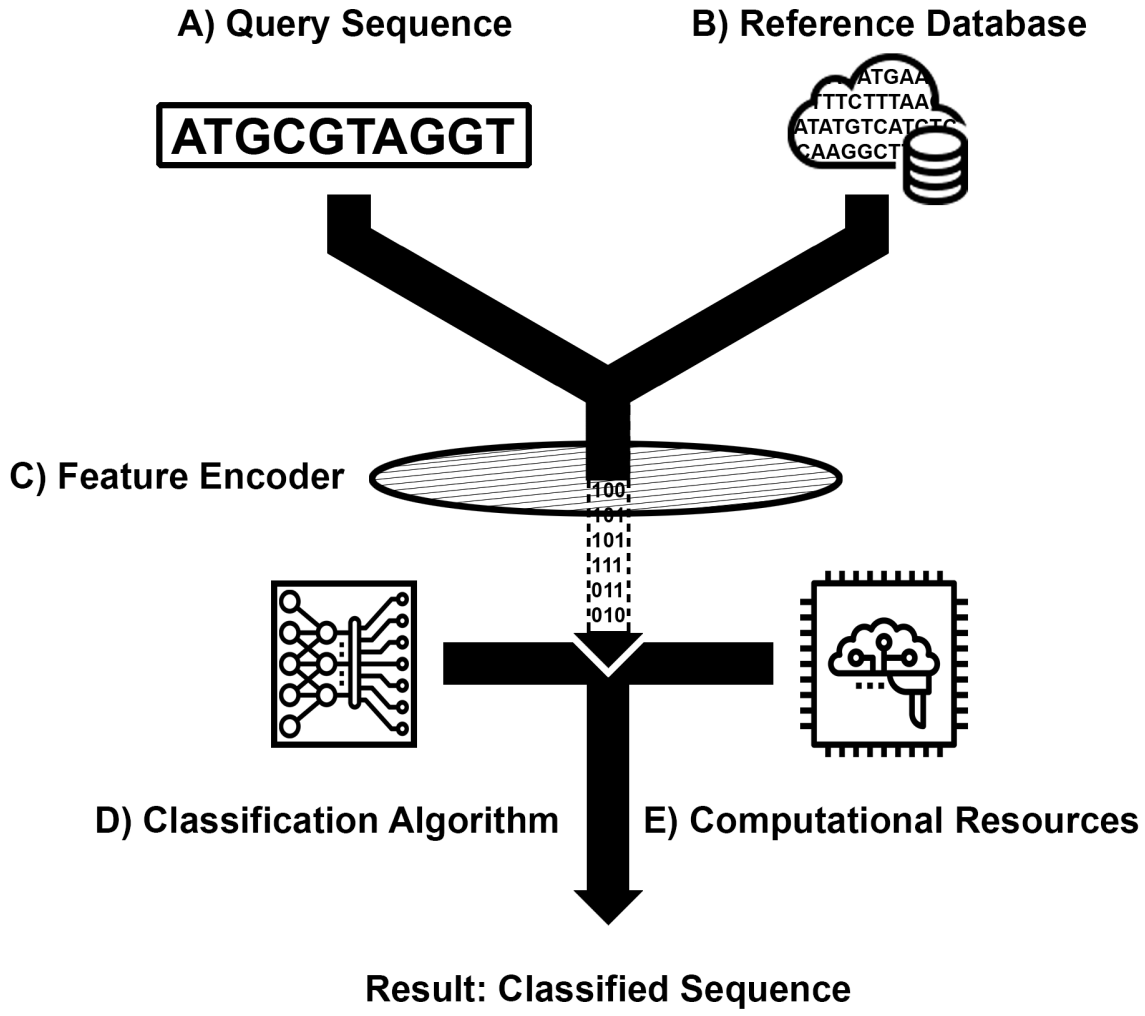

Figure 2.1: Genetic sequence classification is composed of five important areas. A) Query sequences are the unknown data to be classified; these sequences can be short- or long-read data from the sequencer or assembled/aligned sequences that require annotation and classification. B) Reference databases are used as training or known datasets during the process of classifying the query sequence data. C) Nucleotides must be converted into computer-readable information for analysis. The goal of feature encoding is to free otherwise hidden information in the nucleic acid sequences such that the downstream algorithm can capture that important information. D) The classification algorithm determines how the known labels from the database are transferred to the unknown query sequences; algorithms perform the actual classification step. E) Access to computational resources influences which methods are available to the analyst from areas A-D.

gies must be chosen to fit the classification task and the computational resources available to the analyst. Taken together, these results have implications for how genetic sequence classification is performed, particularly for the task of surveillance of high-consequence pathogens or antimicrobial resistance genes related to public health initiatives, since these classification tasks often involve the sensitive detection of genetically variable sequences. This work also provides guidance regarding the development of future classification strategies, including the creation of more robust genetic sequence databases, feature encoding methods, and algorithmic approaches.

2.4 Results and Discussion

2.4.1 High quality and long query sequences provide the best information for genetic sequence classification

Query sequences are the unidentified input data that the analyst desires to classify. These can be short- or long-read data produced directly by sequencing platforms. They can also be assembled sequences that require annotation or identification. The quality of these sequences is important for accurate classification; the fewer errors introduced into the query sequence by molecular preparation and the sequencing process, the easier it is for genetic classifiers to correctly match them to known reference sequences. This is why high-fidelity sequencing platforms like Illumina have historically been more commonly used than sequencing platforms with high error rates [18, 19]. However, the length of the query sequences also affects the classification accuracy, and in some cases, longer query sequences may be more desirable than high quality, short sequences [20]. Query sequence length is important because not all sequence regions are equally informative. Repetitive regions and regions that match to many target reference sequences contain less discriminatory information than unique and long sequences that span repetitive regions. These considerations are analogous to designing polymerase chain reaction (PCR) primers: the region amplified by the PCR process is more discriminatory if that region is unique to a specific genome or if that region is very long and can span repetitive genomic sequences [21, 22]. Therefore, the

ideal query data for sequence classification are as long as possible, high quality (with low error rate), and spanning informative genetic regions.

The characteristics stated above can be influenced by the choice of sequencing platform and molecular preparations used, and we point the reader to other work for further discussion of these points [23–25]. For the purpose of genetic sequence classification, it is critical to understand that the quality of the query data is of the utmost importance: poor quality input data will produce poor quality results, even if the best possible database and algorithm are used to perform the classification task. Therefore, the method by which the query sequences are produced must be as carefully considered as the choice of downstream methods used by the analyst.

2.4.2 Robust databases are the cornerstone of genetic sequence classification

The other input data required to perform genetic sequence classification are labelled reference sequences, often packaged into a reference sequence database. It is our goal in this section to emphasize just how critical the choice of a database is for genetic sequence classification and to generate discussion on the importance of robust sequence databases going forward. Though we recognize that some classification or clustering tasks may be unsupervised (not using labelled data) [26, 27], the majority of analyses that seek to understand genetic data involve supervised classification. We therefore focus on supervised classification in this work.

The task of assigning labels always relies on some known information, which commonly comes from a curated genomic database like those hosted by the National Center for Biotechnology Information (NCBI) or from a curated gene database like MEGARes, which we use in this work [17]. Because labeling is required for most analyses and affects all subsequent analyses, it is crucial that the database used to perform this step is of the highest possible quality and relevance to the analysis being performed. In this section, we discuss how databases are organized and used in genomic classification and propose guidelines for the creation of robust genetic sequence databases.

Though not required, most databases have an organizational structure (annotation) imposed over the genetic sequences contained within them. This structure is known as an ontology: the

ontology typically includes metadata about the database sequences and defines relations between sequences. For example, in the MEGARes database, there are four levels of ontology arranged in a hierarchical structure: Type, Class, Mechanism, and Group. A sequence belongs to a Group, which in turn belongs to a parent Mechanism, the Mechanism to a parent Class, and so on; for example, the sequence entry with accession MEG_1786 belongs to CMY (Group), Class C beta-lactamases (Mechanism), beta-lactams (Class), and Drugs (Type). Multiple sequence entries belong to each ontological category, and these sequences are often biologically or genetically related. For the purpose of genetic sequence classification, the sequences contained within each ontological category must be genetically related to within a certain degree of nucleotide similarity for classification algorithms to function appropriately. This is particularly true of classifiers that rely on k-mer sequence encoding to produce the feature space, as discussed below. Because performance of genetic classifiers is sensitive to database organization and integrity, it is vital that databases be as robust and representative as possible.

Here, we refer to a "robust" database used in genetic sequence classification as one that adheres closely to the following standards: 1. the genetic variation contained in the database should be representative of the genetic sequences being classified against it; 2. the database ontology should group sequences into genetically similar categories while still preserving biologically relevant metadata; 3. the database should be structured and developed to suit the chosen analytic task and classification algorithm, where applicable; and 4. the database annotations should be manually reviewed and linked to primary literature sources, where possible, such that the sequence labels are accurate and verifiable. Below, we discuss each of these points and how they affect genetic sequence classification, pulling in relevant results from this and other work where applicable.

First, consider standard 1 of database robustness introduced above: "the genetic variation contained in the database should be representative of the genetic sequences being classified against it." Database ontological categories typically contain genetically similar sequences and some variation therein: for example, the MEGARes database CMY Group contains a curated set of different CMY beta-lactamase antimicrobial resistance genes, which are within 80% genetic similarity of one an-

other as determined by the CD-HIT clustering algorithm [28]. Therefore, the sequences in this ontological category form a representative genetic profile for the CMY beta-lactamases with some amount of genetic variation around that core genetic profile. One can envision this as a "cloud" of genetic variation centered around a point in genetic space.

When an unknown query sequence is being classified, the classifier determines whether the query is genetically similar enough to the CMY genetic cloud to label it as belonging to the CMY category. The query is only allowed to be so far away from the CMY genetic cloud before the algorithm determines that it does not belong to the CMY category. This tolerance is dependent on the analyst's choice of classifier and the classifier's parameters. If the query is allowed to be classified at too far a genetic distance away from the CMY category sequences, it can result in misclassifications (false positives), since the analyst does not know *a priori* whether the query sequence is truly a CMY sequence or not. Therefore, the ideal choice is to set strict (small) classification distance thresholds, only allowing class assignment when the query sequence is genetically similar to the database to improve accuracy. However, this also has a cost: if the database is not robust according to standard 1 (and is therefore not representative), a phenotypically true CMY sequence could be absent from the database, causing a strict genetic classifier to falsely reject the query sequence from the CMY category. This would result in an increase in false negatives (either misclassifications or "unclassified" reads); and since false negatives are difficult to detect in benchmarking and in practice [29], this problem highlights the need for genetically representative databases.

The issue of high false negative and "unclassified" rates in genetic sequence classification due to non-robust databases is extremely common and not often discussed in current literature. Evidence of this can be found in many -omics fields but is particularly noticeable in the metagenomic sequencing of non-human microbial niches [30]. In Stewart et al. 2019, a suite of classification algorithms was used to perform metagenomic microbiome classification on samples collected from cattle and sheep rumen fluid [31]. Even utilizing a comprehensive suite of classifiers, only 15% of the assembled contigs were classifiable against known reference genomes. However, the authors addressed this issue by using a combination of short-read (Illumina), long-read

(Nanopore), and augmented (Hi-C) sequencing strategies to produce a set of high-quality, *de novo* assembled genomes from the cattle rumen. They then augmented existing databases with these novel genomes, which improved their classification rates from 15% to 70%. Some of these novel genomes were identified as closely related variants of known microbial species, while the majority were only remotely related to known organisms. Furthermore, when this genome-augmented database was then applied to sheep rumen metagenomic analysis as opposed to cattle, the classification rate again dropped from roughly 70% to 50%. These results, in spite of using a comprehensive suite of state-of-the-art classifiers, highlight the critical nature of robust, representative databases for genomic sequence classification.

It is easy to find examples of non-robust databases in microbial metagenomics since the classification objective is so broad; namely, to identify all possible microorganisms. However, this same issue of database non-robustness can also be present in classification tasks of a narrower scope. In this work, we explore this issue for targeted classification of antimicrobial resistance genes using the MEGARes v2 database. As seen in the Stewart et al. 2019 study, unclassified genetic sequences can be one of: 1) closely related to known sequences, 2) extremely distant from the database, or, 3) exist on a continuum between these extremes. It is impossible to know *a priori* which genetic classes in the database will have phenotypically relevant (antimicrobial resistant) variants that are closely related (highly conserved) or genetically distant (highly variable) to the query data. However, it is this problem – the identification of where our databases might be non-robust – that can benefit from the use of a suite of different classification algorithms. We will showcase how the use of multiple classifiers can help detect issues with database robustness in Section 2.4.4.

In addition to databases being representative, it is important for robust databases to be structured appropriately for genetic classification to succeed. Consider standard 2 of database robustness as outlined above: "the database ontology should group sequences into genetically similar categories while still preserving biologically relevant metadata." In database construction, there is a delicate tradeoff between biological information and genetic distance: the result of any classification or analysis must be biologically interpretable to be useful, however the genetic sequence

groupings must be mathematically sensible for the classification algorithm to function accurately. This issue is an extension of a similar problem that the biological field has had historically with taxonomy: the way in which we understand and observe biological phenotypes does not have a 1:1 relationship with changes in the genetic code. For example, a single nucleotide polymorphism (SNP) can have drastic effects on genetic translation, protein production, and ultimately, organism behavior, but the genetic change is to only a single nucleotide. Since the algorithms can observe only the genetic sequence encoding and not the phenotype, large phenotypic changes are not always reflected in changes to genetic classifier behavior. Similarly, organisms were historically classified based on observable phenotypes, such as in the field of cladistics [32]. However, as sequencing and other -omics technologies have become more commonplace, our understanding of organismal classification has shifted to accommodate new information about differences in genotype, but such shifts are not always easy to reconcile, particularly when the genotype differences don't correspond to differences in phenotype [33].

Therefore, like taxonomy, database creation involves a balance between grouping based on biological metadata and the genetic makeup of each ontological group. The choices made in the creation of a database ontology directly affect classifier performance, subsequent analysis, and the inferences we are able to make as a result of the classification task. Ontological groupings that favor biological interpretation over genetic distance can have drastic effects on classification performance. For example, in this work, we utilized the MEGARes v2 database, whose ontology was originally created to respect this phenotypic/genotypic balance [17]. However, while producing results for this work, we noticed a roughly 5% error rate in classification performance across all algorithms against simulated data identical to the sequences in the MEGARes v2 database. When classifying query sequences that are identical to the database in supervised classification, the error rate should be close to 0%, since the query sequences match exactly to the supervised dataset used to inform the classification task. All misclassification error under these circumstances is due to feature overlap in the database ontology, i.e. two different groups, like SHV beta-lactamases and TEM beta-lactamases, share regions of genetic homology such that the classifier cannot decide

to which class the query sequence belongs. This occurs either when the query sequence resides entirely within a region of sequence homology shared between two or more ontological groups in the database or when database sequences are incorrectly annotated, resulting in misclassification (Figure 2.2).

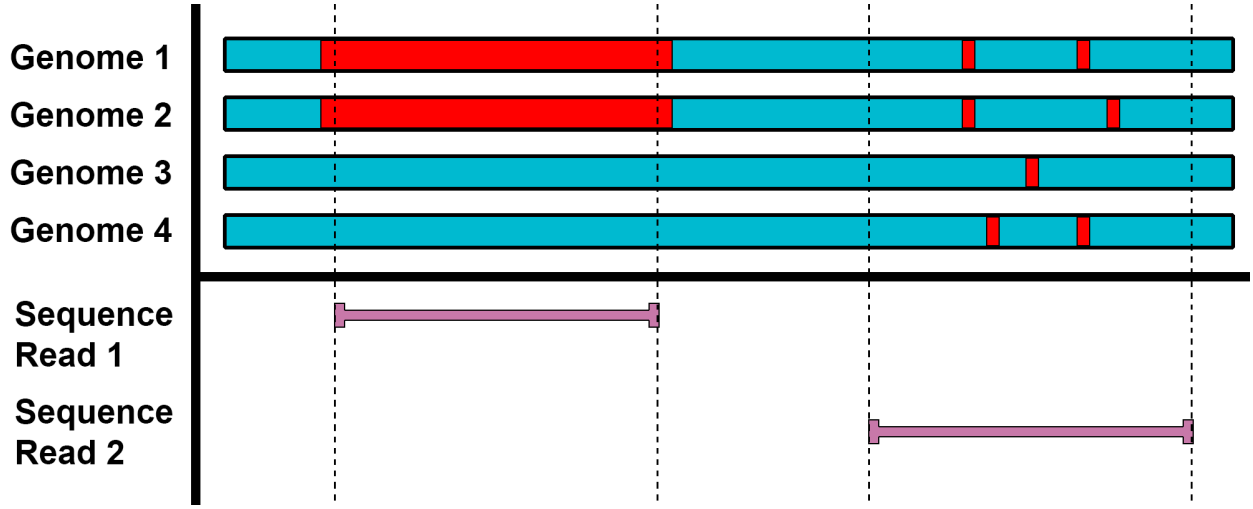

Figure 2.2: Sequences are discriminatory if they span regions of difference between genomes. Sequence read 1 is contained entirely within a region of homology between genomes 1 and 2; it can therefore discriminate genomes 1 and 2 from 3 and 4, but it cannot discriminate genome 1 from genome 2. Sequence read 2 spans a region of high information content, where multiple mutations discriminate between all four genomes. Sequence read 2 is therefore discriminatory and contains a high degree of information.

The first type of misclassification error – misclassification due to homology – is an issue inherent to genetic classification that cannot necessarily be solved by changing the database ontological structure. For example, if the classification task is to assign query sequences to either subspecies *Escherichia coli* K12 or *Escherichia coli* O157:H7, there will be large spans of both genomes that are 100% homologous, even if a number of mutations are present. In fact, only query sequences that span regions that differ between these two organisms can be used to correctly classify them, since query sequences that are contained within homologous regions will be identical for both genomes (Figure 2.2). The sequence regions that differ between these genomes are the only valuable pieces of information for this classification task, a fact that is leveraged by an entire category of genetic classifiers that use these areas of genomic differences, often called “marker genes” (e.g.

Centrifuge, MetaPhlAn) [34, 35]. While misclassification errors due to homology between ontological categories of the database can be minimized by appropriate ontological structuring and classifier algorithm design, the analyst will likely encounter some amount of homologous misclassification error if the database being used for classification is sufficiently large and database classes contain some ontological overlap. This kind of error is also why, for example, many microbiome classification results can only be trusted when classified at the genus level or higher in the taxonomic tree, unless the database and algorithm are specifically designed to differentiate at the species level or lower.

The second type of misclassification error is due to database sequence misannotation or inappropriate ontological structuring. In this case, a sequence in the database that truly belongs in one group is mislabeled as belonging to an inappropriate group. This causes artificial overlap of sequences, and therefore overlap of features (see Section 2.4.3), between two groups. This kind of misclassification error is particularly pronounced for classifiers that use a k-mer feature space, since the confidence in a given classification using k-mers is typically tied to the number of k-mers that match a given ontological group. When a group contains sequence members from its own group as well as another unrelated group, it gains the features of both groups, which results in misclassification to that group at a higher rate than would occur without the database misannotation.

We determined that this scenario was causing the majority of the 5% classification error discussed above with regard to the MEGARes v2 database: several sequences had been misannotated to incorrect ontological categories, therefore some ontological categories required merging due to high amounts of genetic overlap. This kind of error is not surprising, since databases are frequently updated, and some errors in annotation can be made at the time of new data inclusion. The misannotation errors we identified were reported to the database curators and have since been corrected; we also provide for comparison the original MEGARes v2 database and annotation files alongside the modified files used in this work, which are available in the publication GitHub repository (see Methods). The second correction we made was to merge some ontological groups that were too genetically similar for the classifiers examined in this work to separate them. This is an inten-

tional design choice that database curators or sequence analysts must make, as there is no standard practice for how to organize ontological categories over genetic databases. In fact, the database ontology can (and at times should) be modified to suit the analysis being performed, since different algorithms and feature encodings require different genetic distances between groups to perform well.

This leads to standard 3 of database robustness as outlined above: "the database should be structured and developed to suit the chosen analytic task and classification algorithm, where applicable." Ontologies have different structures that convey information about different kinds of relationships between the ontological categories. One can think of ontological structures as mathematical graphs: graphs contain nodes, which represent the ontological categories, and edges, which represent the relationships between nodes (Figure 2.3a) [36]. If one node has a direct relationship with another node, there exists an edge between them. Edges can be directed, which indicates flow of information, or undirected, which simply indicates a nondirectional association between the nodes. Ontological graphs can also be structured in a hierarchy with multiple levels, similar to the taxonomic tree of life (Figure 2.3b). Graphs can be cyclical, such as in metabolic or gene regulatory pathways (Figure 2.3c-d), or acyclical, such as in the taxonomic tree of life.

In the context of genetic sequence analysis, one might choose a particular ontological structure for population count data analysis and a different ontological structure for metabolic or gene regulatory pathway analysis, since these ontologies convey different kinds of information about the database sequences [37, 38]. The structure of an ontology graph has implications for both genetic sequence classification and for subsequent statistical analyses. For sequence classification, the ontology graph often guides the training of learned parameters for the classification algorithm or is used during the process of classification, if the algorithm is heuristic instead of learned. An example of this is Kraken2, which utilizes the ontology graph in a heuristic manner to determine both classification assignment as well as the level of assignment in the ontology, if the ontology is hierarchical [16].

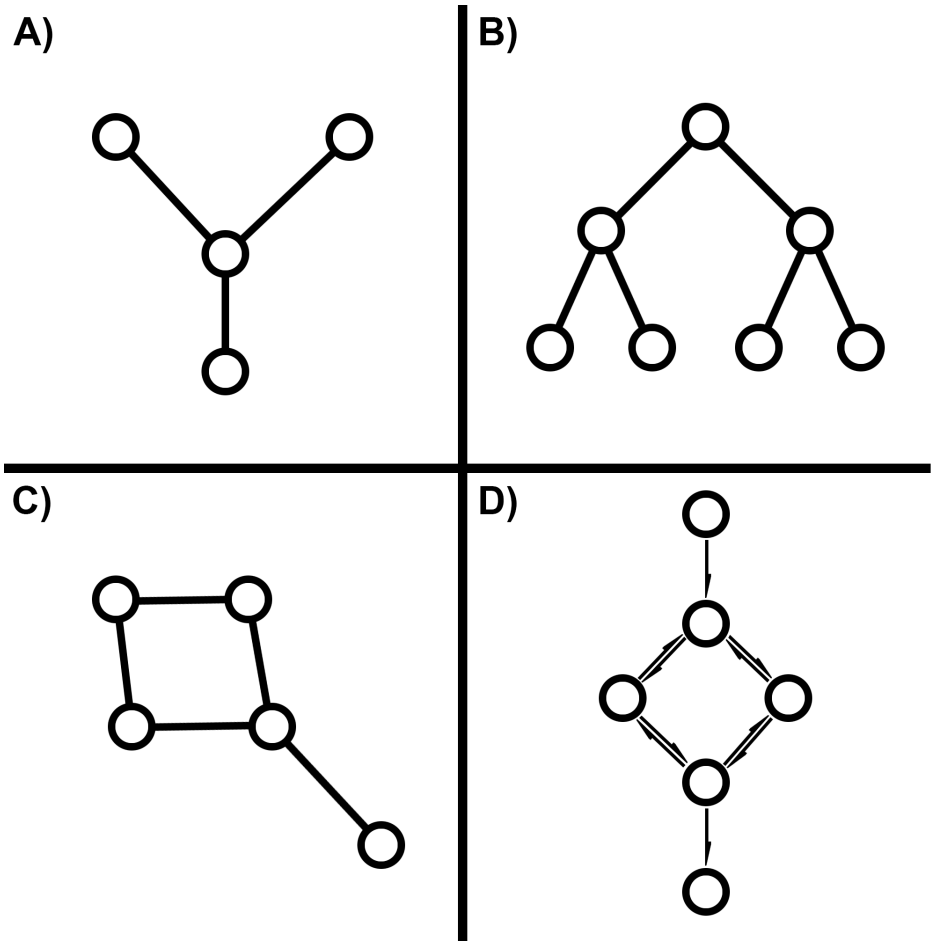

Figure 2.3: Different ontologies define different relationships between ontological categories. A) Ontological graphs are composed of nodes (circles) and edges (lines). Nodes represent database categories, which typically contain genetically related sequences. Edges indicate relationships between nodes, either indicating a flow of information or parent-child relationships. B) A hierarchical graph has levels and contains no cycles. This generally indicates broad groupings at the highest-level nodes, with more specific groupings contained at lower levels; an example is the taxonomic tree of life. C) Graphs can also contain cycles, such as the cycle formed by the four left-most nodes in this graph. D) Cyclical graphs can be directed, indicating a flow of information along the arrows. Directed ontology graphs include metabolic networks and gene pathways.

Database ontological structure can also affect subsequent statistical analyses, after classification has been completed. An example would be an analysis that utilizes classification counts (i.e. counting the number of query sequences that are classified to the database) [39, 40]. This is common in molecular ecology population analyses, where the ontology graph is usually hierarchical (e.g. the taxonomic tree of life). The acyclical ontology graph is a deliberate choice for this analysis, since if there were cycles in the graph, the cycles would introduce conditional dependencies over nodes that are members of a cycle. For example, for every parent node in an acyclical graph, the total count of the parent node is equal to the sum of counts in its children nodes. If there were cycles in the graph, this wouldn't necessarily be true, since when propagating counts up through levels of the graph, counts may be duplicated if they are propagated up through nodes that contain multiple incoming edges (are part of a cycle) (Figure 2.3c). In this way, presence of cycles breaks the hierarchical structure, rendering the graph less useful in count data analysis. Thus, the acyclical ontology graph is used to simplify this process, since nodes on the same level can be considered independent in the acyclical case. It is therefore best to structure the database ontology to match the analysis being performed, when possible, since this will produce the best results for both classification and downstream statistical inference.

Finally, according to standard 4 of robust databases as outlined above: "the database annotations should be manually reviewed and linked to primary literature sources, where possible, such that the sequence labels are accurate and verifiable." With the large number of databases currently in use, it is important to keep track of the origin of database entries and their ontological labels. Early on in the bioinformatics field, computational annotation of sequences was more commonly performed to reduce the workload of manual sequence curation. However, this introduced some misannotation errors into well-known databases, which were subsequently propagated to other, child databases over time [41, 42]. It is therefore important to maintain a record of the original sources for genetic sequences, both to maintain accurate database information and to allow for restructuring of the database ontology and metadata if required.

We acknowledge that these standards for database robustness are not comprehensive and may have limitations in specific circumstances. However, we hope the above discussion of databases and ontologies has convinced the reader that databases are a critical component of genetic sequence classification, perhaps more so than the choice of classification algorithm. Since much of the literature is focused on algorithmic design and performance, we hope that this section will create discussion in the community regarding the need for database development, maintenance, and curation as the field continues to progress. As a final point, we note that fields like natural language processing and computer vision in the broader context of machine learning have focused substantial resources on database creation and maintenance [43, 44]. In some cases, since the databases are now so large in these related fields, classification algorithms are being specifically designed to handle large databases for use as supervised training sets, which has led to development of specialized techniques like transfer learning [45]. Though databases for genetic sequence classification are not yet at this massive scale, there exist initiatives that may produce extremely large sequence databases that require similar efforts in algorithmic efficiency in bioinformatics, and both database curators and algorithmic designers should consider this when planning future research initiatives [46].

2.4.3 Feature encoding determines the information that is available to genetic sequence classifiers

Once query and database data have been provided as inputs, the first step of genetic sequence classification involves encoding the sequence data into a feature space that can be understood by downstream algorithms. The goal of feature encoding is to capture as much information as possible from the genetic sequences, such that the classification algorithms can classify the query sequence to the most appropriate database category. The choice of encoding strategy determines how much of this information is presented to the classifiers. Sometimes, there is sufficient information present in the sequence as is, such that no additional transformation is required to achieve accurate results. Other times, the sequence must be engineered into a different feature space in order for the algo-

rithms to perform well on the classification task. It is difficult to know in advance what kind of feature encoding is required for a given problem, since the classification task depends on all of the factors covered here as well as specifics related to the biological niche under study and the experimental design.

Algorithms designed to operate on genetic sequences, such as alignment, require no transformation of the sequence to compute on them. These algorithms operate on the full-length genetic sequences; however, they often represent sequence nucleotides in a compressed format to improve computer memory efficiency. Though this default form of sequence representation doesn't have a commonly used name, we refer to it here as the "standard" sequence feature space. We introduce the standard encoding here since it is referred to later in Section 2.4.4.

Alternatively, in the context of machine learning, sequences are frequently represented numerically in categorical, vector format, where each element of a fixed-length vector is the integer number of times a feature occurs in the sequence. The natural choice to represent sequences as a fixed-length vector is to break the sequence into overlapping, contiguous subsequences of fixed length k. These subsequences are commonly known as *k*-mers and are a widely used method for mapping sequences into a fixed feature space [47]. With the standard nucleotide alphabet of A, C, G, and T, *k*-mer encoding results in a combinatorial feature space of size 4^k. The transformation of a genetic sequence into such a feature space is a mathematical bijection, such that identical sequences always map to the same, unique vector, and the unique mapping is reversible. While nucleotide subsequences of fixed length are typically referred to as *k*-mers in the literature, protein subsequences can be referred to as n-grams in some contexts [48].

Feature spaces defined over all possible *k*-mers behave in interesting and sometimes unintuitive ways. Intuitively, as k increases, fewer *k*-mers are shared between non-identical sequences (i.e. more *k*-mers are unique to a given sequence) (Figure 2.4a-b). This behavior is similar to increasing the nucleotide length of a primer in the molecular method of polymerase chain reaction (PCR): as the length of the primer increases, so too does its binding specificity. Likewise, as k increases, the combinatorial feature space (state space) grows exponentially at the rate of 4^k which distributes

the counts of *k*-mers more evenly across the categorical feature vector. This can be seen by the decreasing maximum and average count of an example feature vector as k increases (Figure 2.4c). When the value of k is sufficiently large, *k*-mer features have an average count approaching 1, and very few if any *k*-mers are shared between non-identical sequences (Figure 2.4b-c). Finally, as k increases, less of the overall feature space contains non-zero counts, which produces sparse feature vectors at high values of k (Figure 2.4a). The value of k at which the feature vector becomes sparse and approaches an average count of 1 in each non-zero feature category depends on the length and number of sequences being encoded into a given feature vector. Longer sequences and sequence databases with higher nucleotide diversity will require higher values of k to reach feature sparsity; we will further discuss how databases affect the feature space and classification objective in the following sections.

Unintuitively, while feature sparsity and *k*-mer uniqueness might be beneficial in some contexts, it is not clear if feature sparsity always results in an easier classification task. In fact, the performance of dense versus sparse feature vectors likely depends on the type of algorithm chosen to perform the classification. Here, we demonstrated that with smaller values of k (approximately less than 7), an algorithm like t-distributed stochastic neighbor embedding (t-SNE) is able to identify local neighborhoods of similar vectors (Figure 2.5) [49]. However, as the state space and feature sparsity increase with increasing k, this algorithm has a more difficult task in identifying related versus distant vectors, as evidenced by the increasing lack of structure in the resulting embedding at k greater than 6. While the vectors of some classes of sequences are still clustered together, the separation between classes of vectors (between colors in Figure 2.5) is less distinct when k is 13 than when k is 3. This may result in classification tasks where large values of k perform well for one kind of algorithm, such as partial aligners, while small values of k perform better for other kinds of algorithms, like machine learning classifiers. We will further discuss algorithmic strategies for genomic classification in the following sections and relate them back to this idea of feature sparsity.

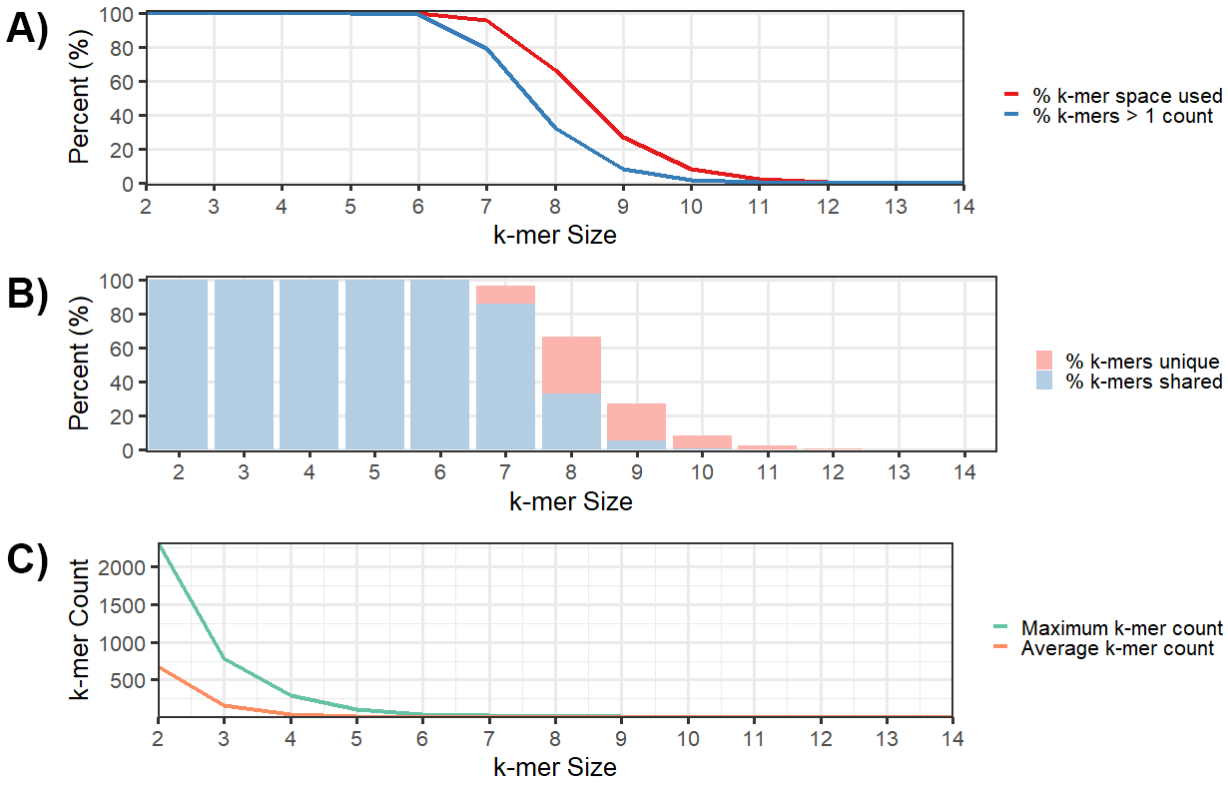

Figure 2.4: Feature spaces using *k*-mers have increasing sparsity and uniqueness between classes as the value of k increases. A) The percent of the k-mer state space with non-zero counts (% k-mer space used) and positive counts (% k-mers > 1 count) decreases as k increases. B) Of the total k-mer state space used (total bar height %), the percent of k-mers shared between classes decreases, and the percent of k-mers unique to one class increases, as the value of k increases. C) The maximum and average count in each feature vector decreases as the value of k increases, eventually approaching a count of 1. For the data used in this figure, 12 AMR mechanisms from the MEGARes2 database were used as classes, 10 sequences were randomly selected from each of these classes, and each sequence was encoded separately into a fixed-length feature vector of length 4^k using standard nucleotide encoding.

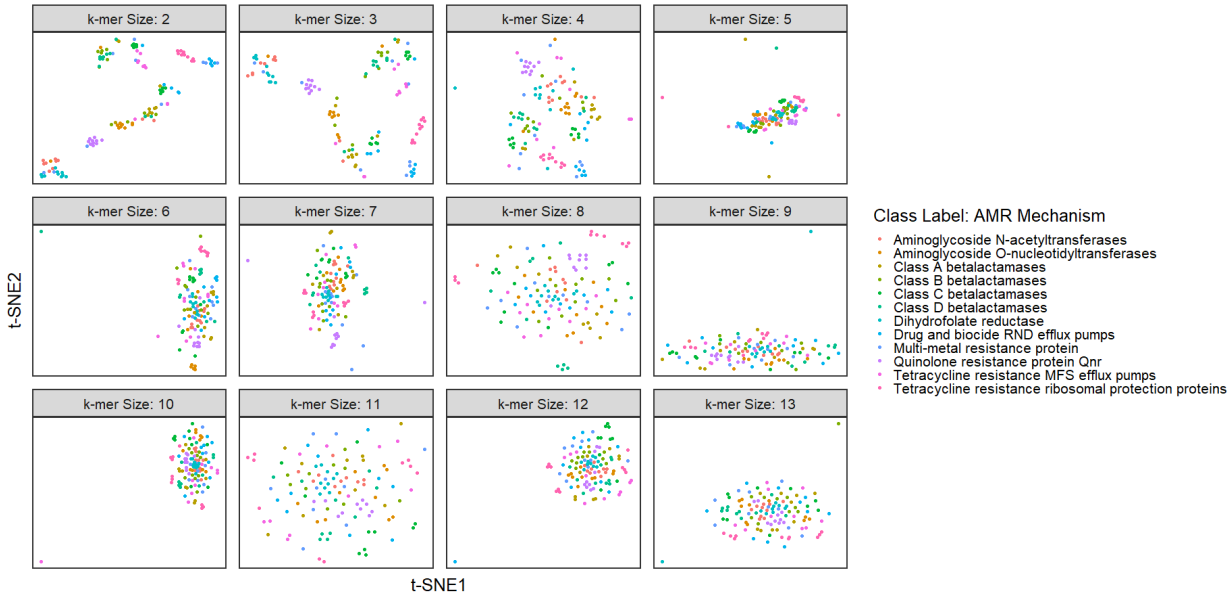

Figure 2.5: Two-dimensional embeddings produced using t-SNE demonstrate different local neighborhoods of *k*-mer feature vectors as the value of k increases. When feature vectors are dense at low values of k, relationships between classes (colors) are well-defined. Alternatively, when k is large, some vectors within classes still cluster together, but the relationship between classes is not as well defined. For the data used in this figure, 12 AMR mechanisms from the MEGARes2 database were used as classes, 10 sequences were randomly selected from each of these classes, and each sequence was encoded separately into a fixed-length feature vector of length 4^k using standard nucleotide encoding.

Clearly, there are advantages and disadvantages to representing sequences in different ways. Using the full-length, standard genomic encoding is still a useful choice for representing sequences, since older sequence alignment algorithms are still commonly used. Unlike *k*-mer representation, full sequence representation allows for the possible identification of mutations and structural variants such as insertions and deletions on the first pass through the data. As a result, full sequence alignment is still used for variant analysis. However, while storing the full length of the query and target sequences is a benefit, it is also a detriment in terms of computational efficiency. Storing and calculating more information requires more computational time and space. Representations using *k*-mers are often more computationally efficient as a result of computing less information, though this depends on the value of k and whether the algorithm computes on the full or a subset of the *k*-mer state space. In further sections, we will focus on how the standard, full sequence representation and the *k*-mer feature representation affect sequence classification, since these are the most widely used methods to represent genetic sequences for the task of supervised classification.

2.4.4 Genetic sequence classification algorithms

Classifier algorithms behave differently depending on their algorithmic class

Once the sequence information has been encoded into an appropriate feature space, an algorithm can be selected to perform the classification task. Our goal in this section is to focus on how major categories of sequence classification algorithms behave in practice. While we provide overviews of the algorithmic strategies for tools showcased in this work, we avoid detailed discussion of algorithm implementations and refer to other sources for further reading. Instead, we tie the choice of algorithm to the points we have already covered about feature spaces and databases, showing how algorithm choice can be utilized by the analyst to ask various questions of genetic sequence datasets. The three major algorithmic categories examined in this work are local alignment, partial alignment, and machine learning (an overview of general machine learning categories with a focus on naive Bayes). Finally, we briefly cover sequential approaches as possible future directions for algorithmic advancement.

Local alignment algorithms have been popular genetic sequence classifiers since the advent of the BLAST (Basic Local Alignment Search Tool) and BLAT (BLAST Like Alignment Tool) approaches [50, 51]. These algorithms are approximate solutions to the problem of global sequence alignment, in which two sequences are compared to minimize the "edit distance" needed to reach one sequence starting from the other, based on some scoring/substitution cost matrix. Global alignment algorithms like Needleman-Wunsch and Smith-Waterman are guaranteed to find an optimal alignment between two sequences, however they are $O(mn)$ in time complexity, where m is the length of the query sequence and n the length of the subject/target sequence [52]. For modern sequence alignment problems, in which billions of sequences must be searched against databases of large genomes, this time complexity is too costly and can result in days or more of compute time. However, they can still be used as gold standard scoring algorithms against which to compare the performance of local alignment algorithms [53].

To avoid the costly time complexity of global alignment, local alignment uses a seed-and-extend strategy, where short *k*-mers (seeds) from the query sequence are matched exactly to the subject sequence and then extended in either direction. If scores remain above a threshold during the extension process, a sequence match is determined, otherwise the seed is rejected. The process of searching for seed matches and extension can be further accelerated by encoding the database in such a way that searching is efficient; this is often achieved by using hash indexing or the Burrows-Wheeler Transform (BWT) followed by a Full-text index in Minute space approach (FM-index) [54, 55]. Once a viable seed and alignment region are identified, a global alignment is performed on the much smaller subregion of the database sequence; this last step is done so that gaps can be inserted where necessary, otherwise the area where the seed first matched would never allow for gaps or mutations, and alignment scores would be suboptimal compared to the gold standard solution. These approaches often result in finding the best possible sequence alignment but are not guaranteed to do so, however they perform much faster on average than global alignment approaches. Due to this compromise between computational efficiency and classification/alignment accuracy, local alignment algorithms have remained a cornerstone of genetic sequence analysis for

decades and are still in wide use. Here, we use the popular software Burrows-Wheeler Aligner (BWA), based on the BWT and FM-index strategies, to showcase how local alignment techniques perform in practice.

Another widely used genetic sequence classifier is Kraken and its variants, due to the extreme speed and accuracy with which they are able to classify genetic sequences [16, 56, 57]. Kraken utilizes exact matching of large, overlapping *k*-mers to perform sequence classification combined with heuristics that consider the feature space and ontology graph. We refer to such large *k*-mer matching strategies as "partial alignment" methods, since the combination of large *k*-mer features and exact matching results in classifier behavior that is somewhere between local alignment and machine learning approaches – a point that we will demonstrate on simulated and real data later in this section. Generally, Kraken encodes both the database and the query sequence into large, overlapping *k*-mers (by default with k greater than 30) then counts the number of *k*-mers from the query that match to a given node in the database ontology. It then uses a heuristic algorithm to determine if the sequence should be classified, to which node it should be classified, and how far down in the ontology hierarchy it can confidently be placed (with ideal placement being as far down as possible, e.g. species level).

Because Kraken does not keep track of where query *k*-mers match in the database sequences and only considers exact matches, it can utilize a very efficient computational data structure called a hash table to perform feature lookups. This data structure has $O(1)$ lookup average time complexity, which is the fastest possible. Additionally, since large *k*-mers are used as features, these *k*-mers are extremely specific to the sequences in the database, making false positive classifications unlikely to occur; this is similar to using a PCR primer of length 32 nucleotides. This results in Kraken being a highly specific (low false positive rate) classifier, though it can result in loss of sensitivity, which is an issue addressed in the more recent Kraken2 release [16]. However, a disadvantage of Kraken is that it only considers the *k*-mers encountered in the database and does not use the entire *k*-mer feature space. With such a large k, it would be impossible to store all 4^k combinatorial features in computer memory. This means that if *k*-mers are encountered in the query dataset

that aren't present in the training dataset, they will not be considered for classification, which has ramifications for detection of mutated sequences. However, in practice, Kraken performs very well despite this drawback, as shown in the results below. Overall, the slightly different approach taken by Kraken results in slightly different classifications than alignment, as seen below in the algorithm comparison discussion.

The last type of algorithm featured in this work is the naive Bayes classifier using small *k*-mers as features [4]. The naive Bayes classifier falls within the broad category of machine learning classifiers, of which there are too many to explore in this work. Success in genetic sequence classification has been had with many kinds of machine learning classifiers, including but not limited to neural networks and their variants, support vector machines, naive Bayes classifiers, and decision trees [4, 58–60]. We chose to focus on the naive Bayes classifier for the following reasons: 1. The algorithm uses the full *k*-mer feature space and therefore makes a good comparison to Kraken, which uses only a subset of the *k*-mer space; 2. the naive Bayes algorithm is sufficiently fast and easily parallelizable, which allows for efficient computation; and 3. naive Bayes classifiers were successfully and broadly used in the past for natural language processing tasks and genetic sequence classification, particularly for smaller problems like 16S gene microbiome work [4, 61]. To explore how this older method compares against modern alignment and partial alignment approaches, we updated this classifier into a modern computational framework called WarpNL, which we make publicly available as a part of this work (see the WarpNL publication code repository under the Supplementary code and data section). Details on the WarpNL design and implementation can be found in the Supplementary Methods.

The combined use of a small k value, the entire *k*-mer feature space, and statistical theory to form a decision boundary further differentiates naive Bayes from alignment and partial alignment. Though machine learning methods are diverse and can produce drastically different results depending on the method, they all rely on statistical optimization and theory to form decision boundaries and perform the classification task. This contrasts with alignment and partial alignment approaches, where the classification decision is ultimately heuristic, even if the heuristic

decisions can be justified post-hoc using statistical confidence metrics. The analyst can leverage this difference to explore an unknown dataset from different angles, since probabilistic methods can provide results that heuristic approaches cannot.

To demonstrate how these algorithmic differences can be leveraged together to explore datasets, we devised two experiments to compare and contrast the algorithms. In these experiments, the MEGARes v2 modified database was used as the supervised/training dataset, and either simulated or gold standard data for AMR sequence classification were used as the query dataset. The goal of the first experiment was to showcase how each algorithmic class handles classification decisions when faced with different kinds of mutational patterns away from the training database. In this first experiment, simulated, single-end, short read (150 bp) data were produced using the MEGARes v2 modified database sequences, and each simulated sequence was injected with a number of mutations ranging from 0 to 15 (10% mutation rate) at either fixed (evenly spaced) or random intervals along the 150 bp simulated reads. We refer to these mutational subsets as the "even" and "random" mutation datasets, respectively. The simulated data were then classified against the MEGARes v2 modified database using the BWA (alignment), Kraken2 (partial alignment), and WarpNL (naive Bayes machine learning) algorithms. Since the goal of this experiment was to demonstrate how these algorithms differ in classifying specific mutational patterns, no additional sequencing error profile was used to generate the simulated data as might be done in a traditional simulated benchmark aimed at determining algorithm performance metrics [62].

Overall, all three algorithms performed well in the face of high mutation rates (upwards of 10% on short read data), with correct classification rates over 95% up to around 5% mutation rate. Beyond approximately 8 mutations per 150 bp (5% mutation rate), the algorithms behaved in different ways and therefore captured different kinds of information about the query data. When mutations were spaced evenly along the simulated read, the machine learning naive Bayes classifier (WarpNL) with small k-mer feature encoding correctly identified more sequences than BWA or Kraken2 as mutation rate increased (Figure 2.6). This is because with a higher number of evenly spaced mutations, the regions of perfect homology to the database (regions between each mutation)

decreased in size. Therefore, use of a small k-mer encoding strategy allowed the algorithm to "see" the regions of homology to the database, where the use of a larger k-mer size would not.

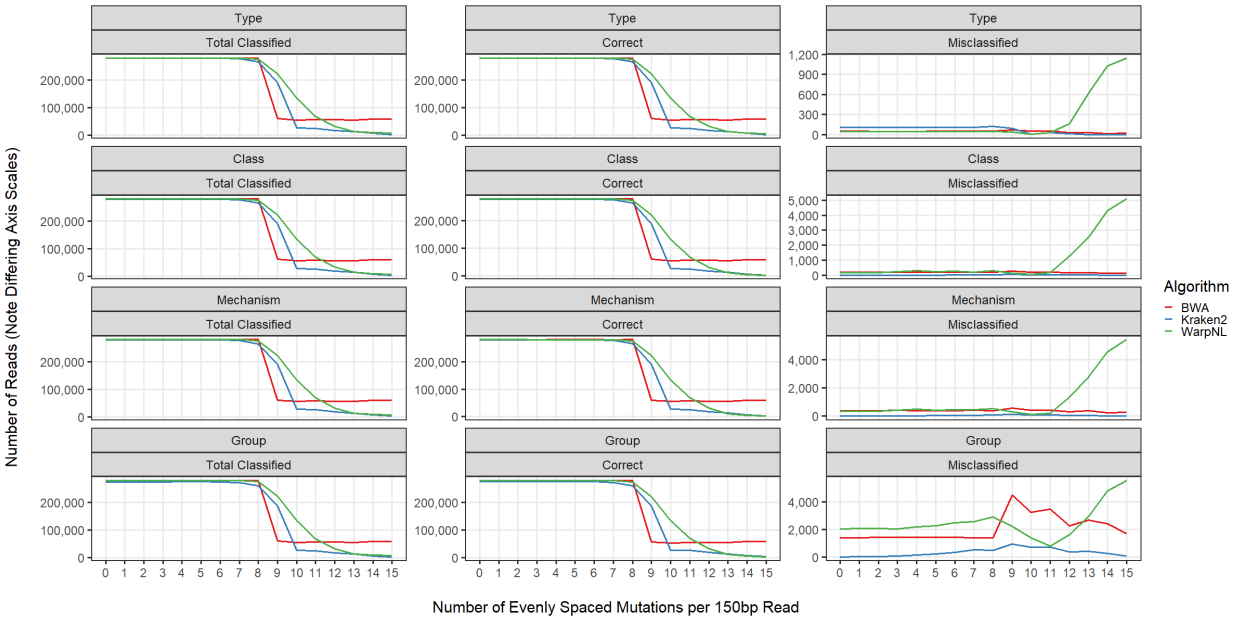

Figure 2.6: When simulated sequences from the MEGARes v2 database are increasingly mutated evenly across the simulated sequence reads, WarpNL correctly classifies the most sequences, followed by Kraken2 and BWA. However, BWA exhibits increased classified rates at very high mutation levels due to recovery strategies implemented in its BWA-MEM re-seeding algorithm. Also, while WarpNL performs best on evenly spaced mutations, it also has the highest misclassification rate at high mutation levels. The rows in this figure are the ontological levels of the MEGARes v2 database from highest (top row) to lowest (bottom row): Type, Class, Mechanism, and Group. The MEGARes ontology graph is a hierarchical tree (acyclical), similar to the taxonomic tree of life. The columns of this figure are the Total Classified reads (true positives and false positives), Correct classified reads (true positives), and Misclassified reads (false positives) by algorithm. The x-axis is the number of mutations per simulated read, and the y-axis is the total number of reads classified. Note the differences in y-axis values across the figure facets.

Additionally, the way in which we implemented WarpNL considers the entire read as a single classifiable unit, i.e. it classifies the entire read in a single decision as opposed to classifying sub-regions of the read independently. Since WarpNL could detect the small runs of homologous bases due to the small k value, it had enough information aggregated across the entire read to correctly classify reads with evenly spaced mutations. However, as the number of mutations increased, this naive Bayes machine learning classifier had the largest increase in false positive classifications (misclassified). One explanation for this is that the mutation rate across the read was high enough

that the algorithm determined that it should classify the read to something in the database, but the read was divergent enough from its true ontological group that the algorithm mistook it for something else. In this particular case, WarpNL had a better sensitivity of detection compared to heuristic methods; however, this gain in sensitivity came at the cost of reduced specificity, which is a well-known tradeoff in classification [63].

On the other side of the algorithmic spectrum, the local alignment technique (BWA) had trouble with the even mutation dataset: BWA demonstrated the steepest decline in correct classification rate as the number of mutations increased (Figure 2.7). This was due to the way in which alignment is performed via the seed-and-extend approach (described above). The small *k*-mer seed matches to a region of perfect homology to the database and then attempts to extend in either direction, however if it encounters too many mutations during the extension step, it will fail to align the read. Since these evenly mutated reads had frequent mutations, the extension portion of the algorithm considered those evenly spaced mutations to be too divergent from the database, causing it to be unclassified.

While BWA had the steepest decline in correct classification rate on the even mutation dataset, it appeared to recover at higher levels of mutation (greater than 9 per 150 bp), resulting in the most correctly classified reads at very high mutation rates. This is an interesting result that showcases how heuristic algorithms can adapt to various circumstances, for example, how the BWA-MEM algorithm was implemented to accommodate increasing read lengths as early as 2013 [64]. In the 2013 update, the developers introduced several logical recovery branches to the algorithm, including a re-seeding strategy that searches for seeds using smaller-than-default k values and additionally evaluates chains of seed matches in local alignment neighborhoods [64]. This was intended to accommodate structural variations that were anticipated to become more common in sequencing data as read length increased, however it evidently was also able to detect the evenly spaced mutational pattern present in this dataset. This result demonstrates an advantage of heuristic approaches over machine learning ones, since fewer stand-alone methods in the field of machine

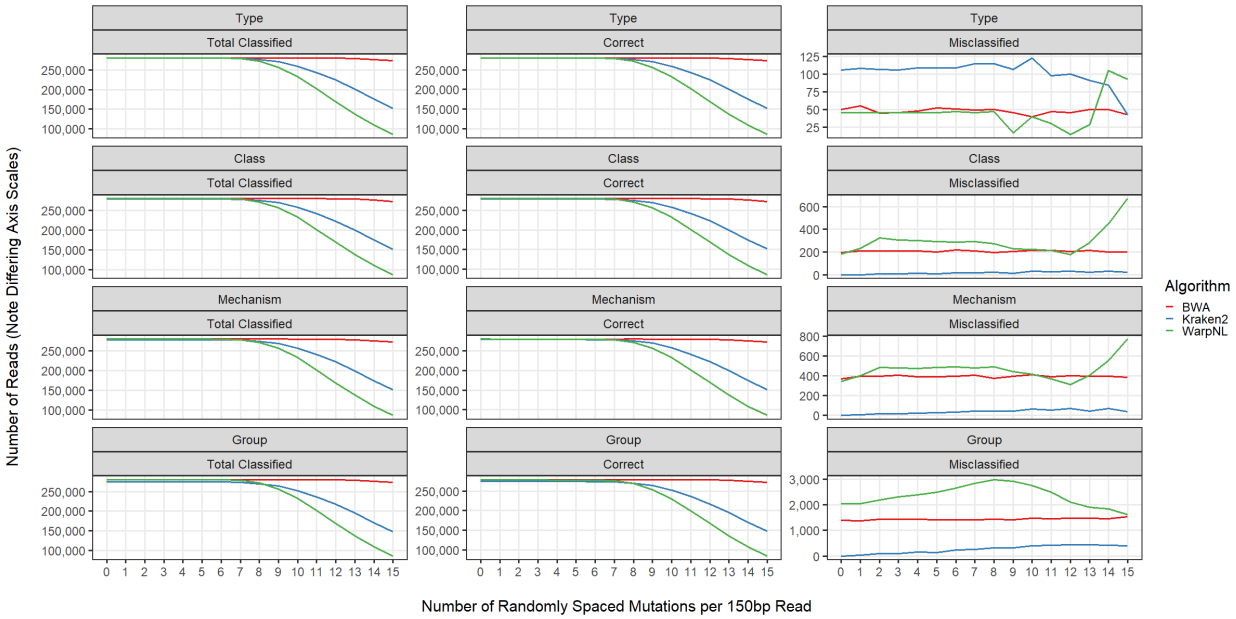

Figure 2.7: When simulated sequences from the MEGARes v2 database are increasingly mutated randomly across the simulated sequence reads, BWA correctly classifies the most sequences, followed by Kraken2 and WarpNL. All classifiers classify greater than 35% of the simulated reads, even at high mutation levels. Note that Kraken2 has the lowest misclassification rate due to its use of large *k*-mers, while WarpNL continues to have the highest misclassification rate. The rows in this figure are the ontological levels of the MEGARes v2 database from highest (top row) to lowest (bottom row): Type, Class, Mechanism, and Group. The MEGARes ontology graph is a hierarchical tree (acyclical), similar to the taxonomic tree of life. The columns of this figure are the Total Classified reads (true positives and false positives), Correct classified reads (true positives), and Misclassified reads (false positives) by algorithm. The x-axis is the number of mutations per simulated read, and the y-axis is the total number of reads classified. Note the differences in y-axis values across the figure facets.

learning would be able to accommodate this kind of logical branching, perhaps with the exception of decision trees.

Kraken2, which we consider here to be a "partial alignment" approach, fell in between the results of local alignment (BWA) and naive Bayes (WarpNL). Kraken2 has advantages of both alignment and machine learning methods: it uses a *k*-mer feature space, which allows it to find smaller regions of homology in larger sequences, but it also has a high specificity in the face of increasing mutations. This is interesting, since having a high specificity is more characteristic of methods like alignment that utilize the full sequence space as opposed to *k*-mers. Additionally, Kraken2 is orders of magnitude faster than BWA or WarpNL when used to classify large datasets, which can compensate for its slight decrease in overall accuracy relative to alignment approaches.

An interesting point is that Kraken2 performs its classification in the context of the database hierarchical ontology graph: note that in these experiments Kraken2 did not classify all reads down to the lowest ontological level (MEGARes Group level) even at a mutation rate of 0 (Figure 2.6). Instead, Kraken2 classifies query data to whichever ontological level its heuristic algorithm determines is discriminatory for that particular ontological branch. For instance, some of the simulated reads generated in these datasets were contained entirely within regions of homology that are shared between two ontological categories. This caused Kraken2 to find the next highest parent node in the ontology graph where nodes within the same hierarchical level did not have too much feature overlap. Since Kraken2 utilizes a large k size (greater than 30 by default), it often takes only one such region of 30-35 bp to be discriminatory, so as long as one such discriminatory *k*-mer is present in the query sequence, Kraken2 can classify it to a lower level. However, if no discriminatory *k*-mer is present, Kraken2 defaults to classification at a higher ontological level. This makes Kraken2 well-suited to classification tasks that involve hierarchical ontology graphs, such as microbiome classification. Also note that Kraken2 is conservative (strict) with its classifications without sacrificing too much sensitivity: it had by far the lowest misclassification rate on simulated data of the classifiers examined here.

Interestingly, the patterns of classification performance seen in the even mutation dataset were reversed for the algorithms when mutations were instead randomly distributed across the simulated reads (Figure 2.7). Random patterns of mutation are more likely to be encountered in real sequence data, since the overall forward mutation rate is greater than the rate of structural variations like insertion/deletion events or mutations in linkage disequilibrium in most organisms [65]. Here, we see that local alignment was robust to random mutation patterns even at very high rates of mutation (10%). The k-mer classifiers, however, were less accurate when faced with a random mutation pattern: the naive Bayes machine learning method (WarpNL) classified fewer than 50% of reads as the number of mutations approached 15 per 150 bp, while local alignment (BWA) classified greater than 95% of reads, and partial alignment (Kraken2) again fell in between. The misclassification rates showed the same pattern in the random mutation dataset as in the even mutation dataset, with Kraken2 having the lowest misclassification rate at the lower hierarchical ontology levels and WarpNL having the highest misclassification rates overall (Figure 2.7).

Taken together, these results demonstrate that different algorithms behave differently, even when used in a classification task with the same query data and database. While this may seem concerning, since there ought to be only one ground truth result for classification, the reality of genetic sequence analysis is that the "ground truth" label for a given sequence depends on many factors, e.g. the biological system in which the experiment is performed, the phenotypic differences the analyst wishes to explore, and the overall goal of experimental inference that the analyst seeks to achieve. It is therefore up to the analyst to use appropriate downstream statistical methodology and subject matter expertise to make well-informed inferences using classification results for a particular genetic sequence analysis. Since the goal is to be as well-informed as possible about the dataset at hand, the different behavior of these genetic sequence classification algorithms can be leveraged by the analyst to explore datasets from different angles and to gain a breadth of information that may be relevant to downstream inference. In the field of machine learning, groups of classifiers used together to make inferences are often called classifier ensembles and are an established strategy for analysis [66].

To illustrate the benefit of classifier ensembles, we performed a second experiment using two datasets that we consider to be the closest to a real-world gold standard for antimicrobial resistance sequence classification. The query data come from two functional metagenomic datasets produced by the creators of the Resfams antimicrobial resistance protein hidden Markov models: here, we refer to these datasets as Pediatric and Soil [67, 68]. Functional metagenomics provides a phenotypic label over traditionally metagenomic data: cleaved fragments of metagenomic DNA from any sample type are cloned into a phenotypically antimicrobial susceptible bacterial vector like *Escherichia coli* and are functionally overexpressed on a plasmid. The formerly susceptible clones are then grown on agar infused with various kinds of antimicrobial drugs. Therefore, colonies that grow on the antimicrobial-laden agar must contain a metagenomic fragment that confers resistance to at least that particular antimicrobial. Since the metagenomic fragment is overexpressed, it should be detectable via sequencing of the bacterial colony. This provides a target label against which classification strategies can be evaluated: if the resulting genetic sequence data contains an antimicrobial resistance gene that is also present in the reference database, then the classifiers should be able to identify it in high copy number/coverage, provided there is sufficient sequencing depth.

However, one should note that the classifiers may also identify off-target genes that might be present; it is not uncommon for antimicrobial resistance genes to be found in close genomic or plasmid proximity in nature, therefore there may be some samples that are resistant to multiple antibiotic types but have only one phenotypic resistance label [69]. Additionally, the classifiers could identify sequences belonging to housekeeping genes like DNA topoisomerases or ribosomes that may or may not be phenotypically resistant, since these resistance mechanisms are often a result of a few mutations to otherwise ubiquitous bacterial proteins. Because of this, we referred to classifier results as being "on-target" if the label assigned by the classifier matched the antimicrobial drug class in the agar and "off-target" if it did not. However, not all off-target classifications were necessarily incorrect due to the reasoning above.

The metagenomic DNA for the Pediatric dataset was isolated from healthy infant and children fecal samples [67], while the DNA source for the Soil dataset was obtained from agricultural and grassland soil samples [68]. The authors of these studies hypothesized and subsequently showed that known and novel antimicrobial resistance genes could be identified in both datasets, however the human-associated Pediatric dataset likely contains more well-studied antimicrobial resistance genes, and the Soil dataset contains antimicrobial resistance genes that are more divergent from known databases. Though these studies were performed circa 2014, this pattern still holds true even using modern antimicrobial resistance databases. Using these three classifiers spanning the methods of local alignment, partial alignment, and machine learning, we hypothesized that each of these algorithmic classes would identify a different aspect of genetic variation away from the MEGARes database, with the Soil dataset results being more genetically distant than those from the Pediatric dataset.

All three classifiers were able to find sequences matching the functional metagenomic antimicrobial resistance labels in both datasets with a high degree of accuracy (Figure 2.8a, Figure 2.9a). Additionally, all three classifiers captured the same core group of sequences, as evidenced by the high level of agreement on sequence classifications between pairs of classifiers and all three classifiers used together. However, individually, the classifiers also captured sequences that didn't match the functional metagenomic labels and some sequences that the other classifiers did not classify.

In the Pediatric dataset, the *k*-mer classifiers (Kraken and WarpNL) identified more off-target sequences than alignment, but when either *k*-mer classifier was combined with alignment, 100% of reads identified were on-target. For the Pediatric dataset, alignment (BWA) outperformed the *k*-mer classifiers in all categories for identification of on-target sequences, particularly in samples exhibiting phenotypic resistance to the glycopeptide antimicrobial drug class (Figure 2.8b). We hypothesize that this is due to metagenomes related to human subjects being more well-characterized and represented in databases than samples not from human subjects [70]. In the case where the cloud of genetic variation in the query data was well-represented in the training database, both

alignment and large *k*-mer partial alignment strategies performed well, since exact-matching subsequences were highly represented in the query data.

In the Soil dataset, while all classifiers identified mostly the same subset of sequences, alignment (BWA) and the small *k*-mer naive Bayes classifier (WarpNL) individually and together outperformed large *k*-mer partial alignment (Kraken) to a small degree on overall and on-target sequence classification. This was most notable in samples exhibiting phenotypic resistance to the glycopeptide and phenicol antimicrobial drug classes (Figure 2.9b). We again hypothesize that this is due to query data in the Soil dataset being more genetically distant from the sequences represented in the MEGARes v2 database. To examine this hypothesis, we further explored off-target classification patterns for each classifier combination and analyzed alignment metrics that might provide an estimation of genetic distance in the query data relative to the MEGARes v2 database.

Off-target classifications in functional metagenomic data can be either false positive classifications (misclassification errors made by the classification algorithm) or correct classifications of genetic sequences that do not confer resistance to the antimicrobial embedded in the agar during the functional metagenomic experimental process. Given the high level of accuracy displayed by all three classifiers examined here, the majority of the off-target classifications were likely the latter and not due to errors made by the classifiers themselves. Typically, if real false positives/misclassifications are produced, they tend to result in a small number of sequences being classified to a wide variety of disparate database labels. This is why a large portion of misclassification error can be eliminated by removing categories with few (less than 5th quantile) sequence counts as a post-classification filtering strategy during statistical analysis [71, 72]. Here, we show the top 7 most abundant off-target classifications made by each classifier and classifier combination for the Pediatric and Soil datasets (Figure 2.10, Figure 2.11).

In the Pediatric off-target classifications, the Class A beta-lactamase counts are nearly all from a single sample in the Pediatric dataset (SRR961818) that contains a true cblA gene Class A beta-lactamase precursor product (Figure 2.10). Phenotypically, this may or may not be a functional AMR gene, however it was independently identified by all three classifiers with a large number

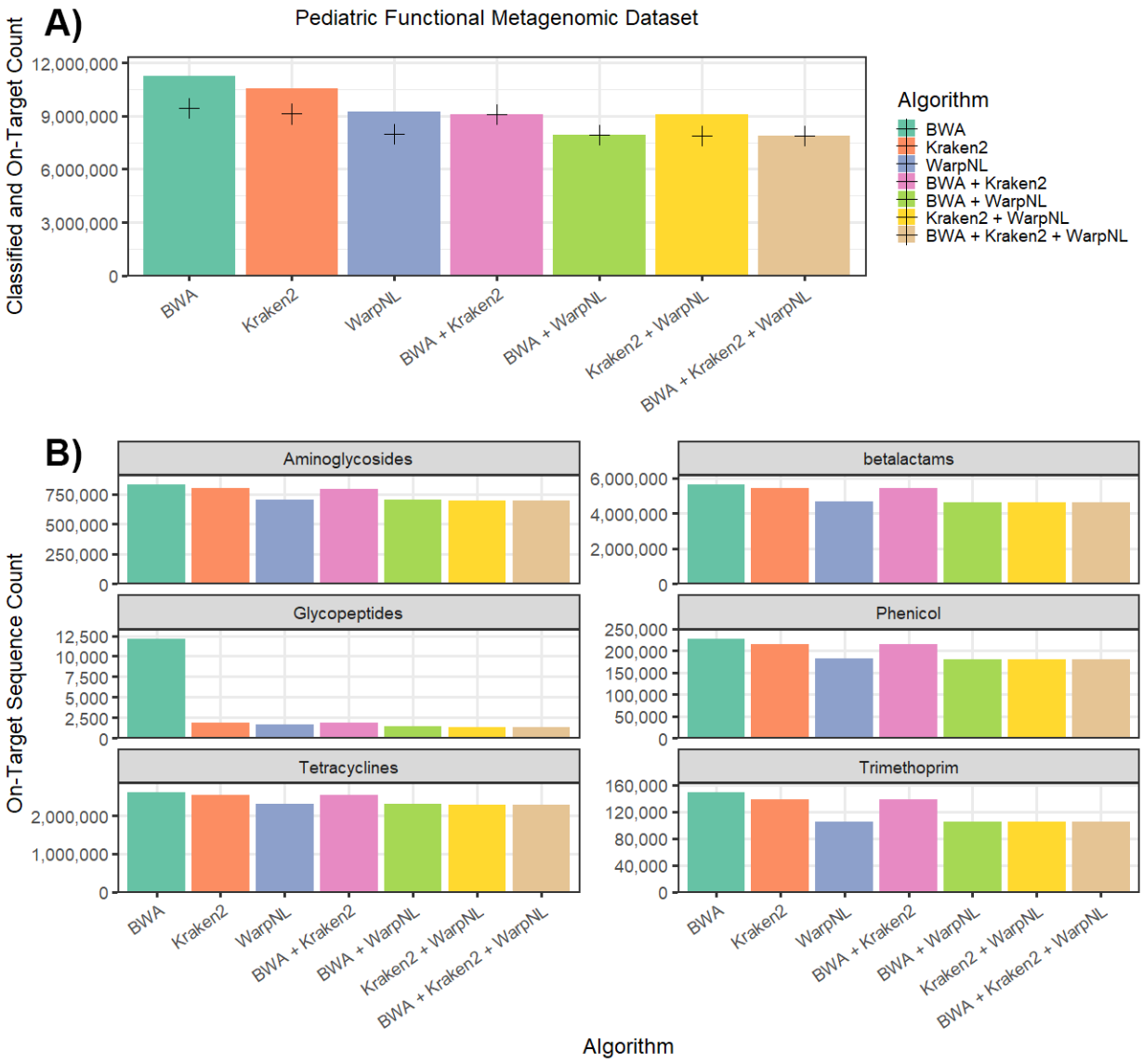

Figure 2.8: In the Pediatric functional metagenomic dataset, individual classifiers identify largely overlapping but unique subsets of sequences, most of which are on-target according to the functional metagenomic phenotypic resistance label. A) The total number of reads classified (bar) and on-target number of reads classified (crosshair) by algorithm (x-axis, color). BWA, Kraken2, and WarpNL identify an overlapping but unique set of query sequences when used individually. In combination, a large percentage of classified sequences are on-target, but fewer overall sequences are identified. The *k*-mer classifiers (Kraken2 + WarpNL) both identify sequences beyond the on-target subset, suggesting that this dataset contains true genetic sequences that match to the database but were not phenotypically labelled as resistant. B) The on-target sequences were subset by functional metagenomic phenotypic resistance label and similarly colored by algorithm. Alignment and partial alignment (BWA + Kraken2) identify the most on-target sequences for the Pediatric dataset. BWA identifies many more on-target sequences belonging to the glycopeptide phenotypic resistance label than do the other classifiers.

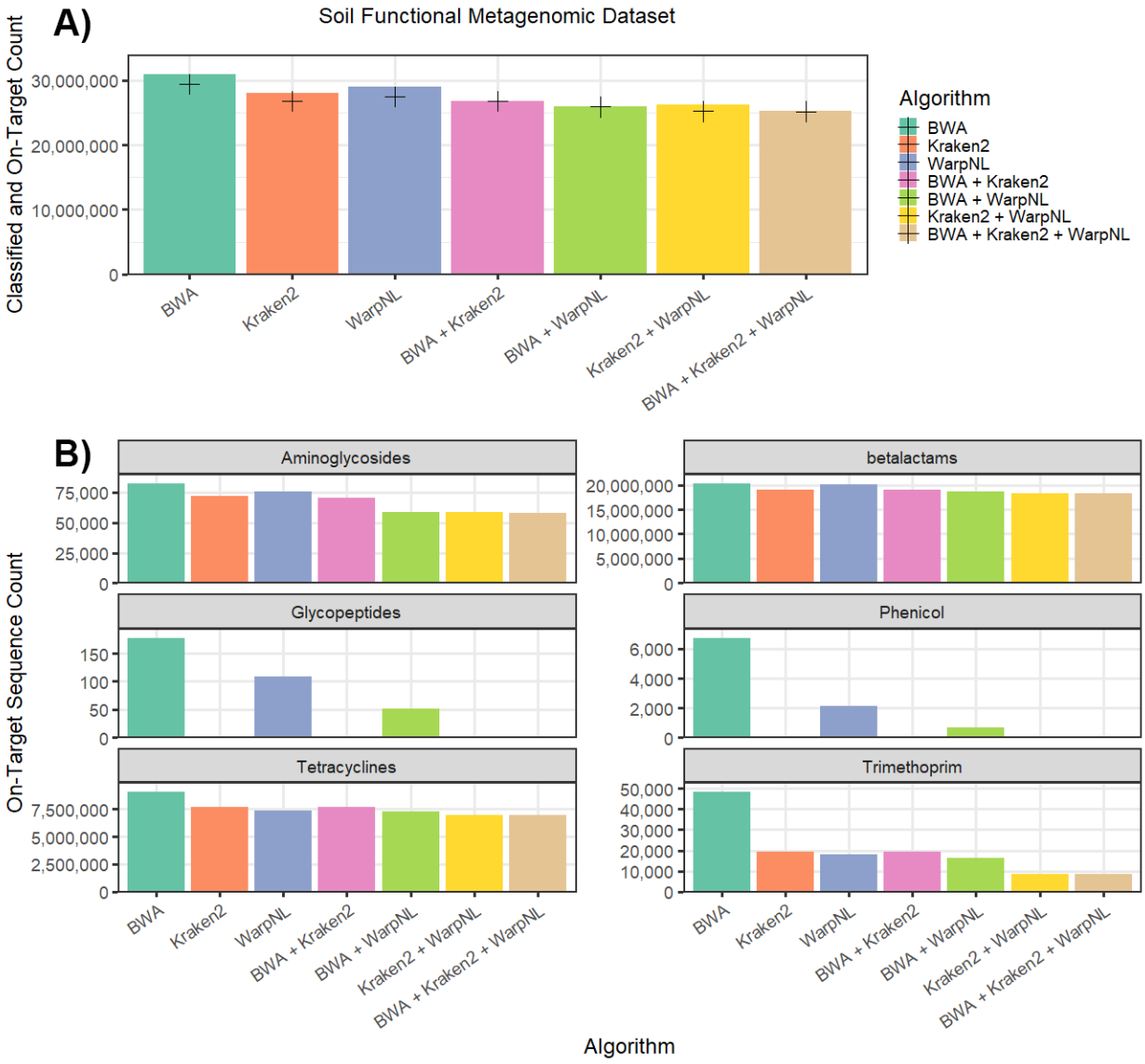

Figure 2.9: In the Soil functional metagenomic dataset, individual classifiers identify largely overlapping but unique subsets of sequences, most of which are on-target according to the functional metagenomic phenotypic resistance label. A) The total number of reads classified (bar) and on-target number of reads classified (crosshair) by algorithm (x-axis, color). BWA, Kraken2, and WarpNL identify an overlapping but unique set of query sequences when used individually, more of which are on-target than in the Pediatric dataset. In combination, a large percentage of classified sequences are on-target, but fewer overall sequences are identified. BWA identifies the most and most on-target sequences in this dataset. Within the *k*-mer classifiers, WarpNL identifies more sequences and on-target sequences than Kraken2 in this dataset. B) The on-target sequences were subset by functional metagenomic phenotypic resistance label and similarly colored by algorithm. Within the glycopeptide and phenicol functional metagenomic phenotypic resistance label, BWA and WarpNL identify some sequences that Kraken2 does not. Additionally, within the glycopeptide and phenicol labels, BWA and WarpNL largely identify non-overlapping query sequences.

(greater than 420,000) of sequences classified. Interestingly, alignment identified a largely different subset of sequence reads belonging to this product than did the *k*-mer classifiers (data not shown). In the following discussion, we will elaborate on why this might be. The remaining off-target categories were various housekeeping genes, including ribosomal subunits (16S and 23S), ribosomal protection proteins, or multi-drug efflux pumps that can be found in many bacterial clades. Again, these may or may not be phenotypically resistant to antimicrobial drugs, however their genetic sequence was similar enough to the sequences present in the MEGARes database such that they were classified. Such off-target classifications require further exploration to confirm the presence of resistance-conferring mutations (usually through *de novo* assembly and protein translation), and many off-target classifications require subsequent wet-lab analyses to verify phenotypic resistance.

In the Soil off-target classifications, nearly all ontological categories identified as off-target belonged to housekeeping genes, similarly to the Pediatric dataset, with the addition of some DNA topoisomerase categories (Figure 2.11). Again, all three classifiers identified a large number of off-target sequences when used individually, however when considered together, the number of off-target classified sequences dropped to below 200 sequence reads per category. This is because the algorithms (in this instance) identified largely non-overlapping sequence subsets, with the *k*-mer classifiers agreeing more with one another than with local alignment. This may suggest that *k*-mer classifiers are, in general, better at detecting sequences that are genetically distant from the database. Yet because *k*-mer classifiers typically don't provide much information about how their classification decisions are made, it can be difficult to identify why this is the case. However, the required information is provided by alignment techniques like BWA, which allowed us to further investigate behavior of the *k*-mer classifiers by comparing the *k*-mer classifiers to the results from BWA's alignments on a read-by-read basis.

Though Kraken2 does provide some information regarding matching *k*-mer location in the query sequence, we chose to use the alignments produced by BWA as a point of comparison for this next experiment. This is because BWA produces Smith-Waterman alignments on a local

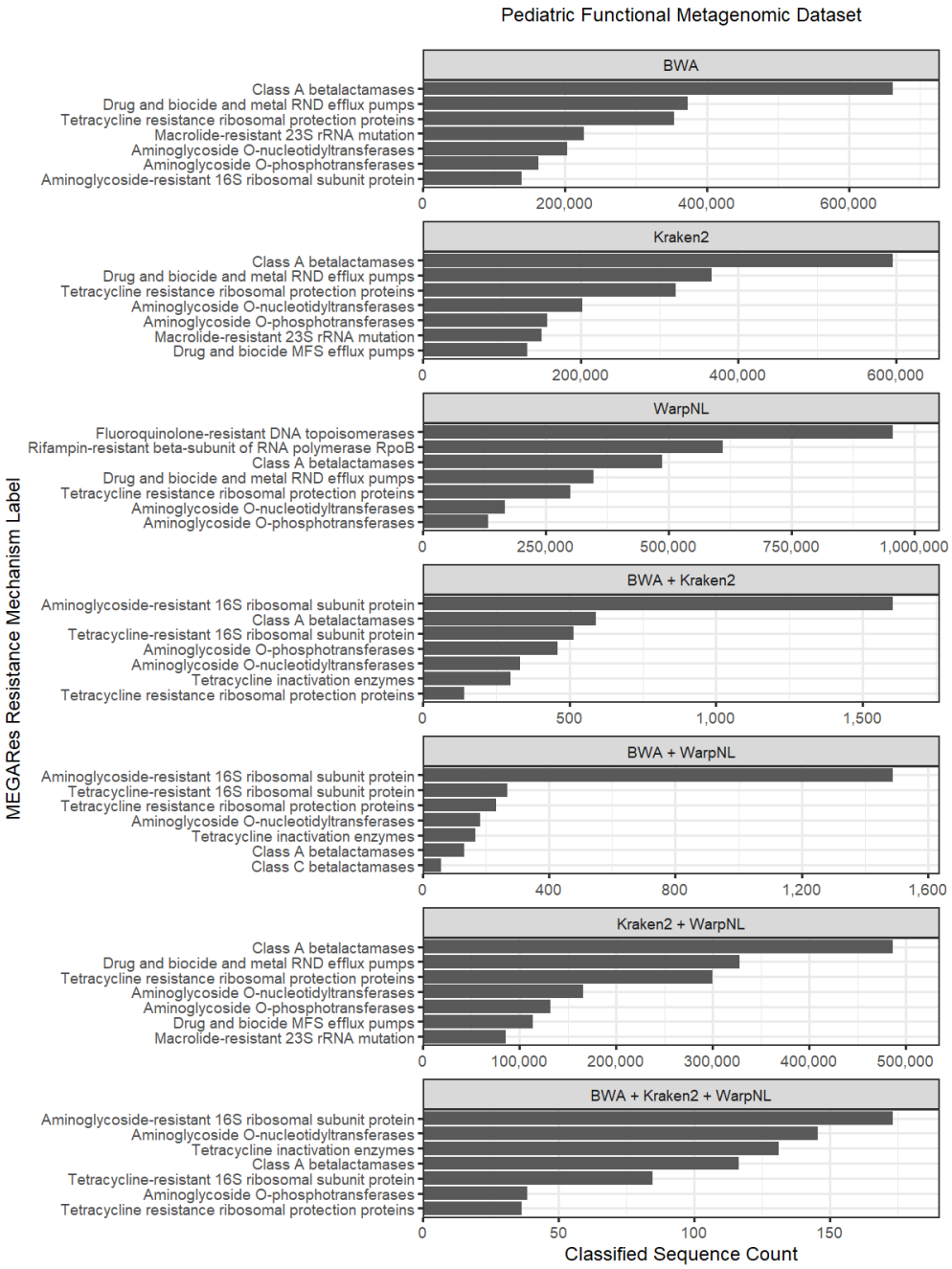

Figure 2.10: Off-target classifications for the Pediatric functional metagenomic dataset are largely housekeeping genes that are genetically similar to the MEGARes v2 database sequences. These sequences may or may not be phenotypically resistant, requiring further bioinformatic or molecular experimentation to verify resistance. All classifiers individually identify a large set of reads belonging to the Class A beta-lactamase ontological group, which is a cblA beta-lactamase precursor from sample SRR961818. However, BWA appears to identify a different subset of sequences than do Kraken + WarpNL, since all three classifiers used in combination result in very few sequences (less than 200) being classified per ontological category.

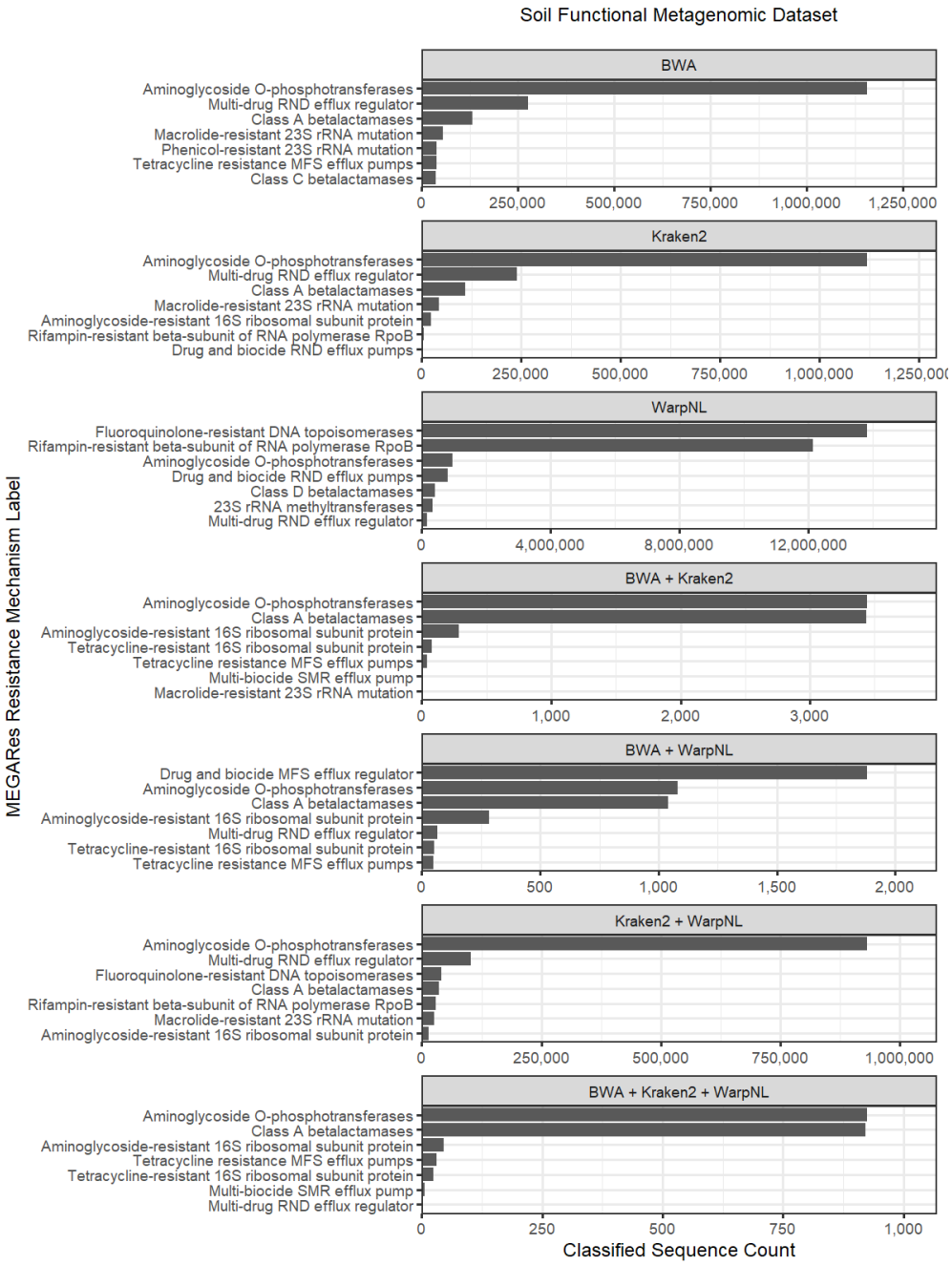

Figure 2.11: Off-target classifications for the Soil functional metagenomic dataset are largely housekeeping genes that are genetically similar to the MEGARes v2 database sequences. These sequences may or may not be phenotypically resistant, requiring further bioinformatic or molecular experimentation to verify resistance. The ontological categories seen in the Soil off-target sequences are similar in context to the Pediatric off-target sequences with the addition of some DNA topoisomerases.

context, which provides detailed information regarding sequence gaps, mutations, and total aligned base pairs from the query sequence against the database sequence. We first extracted the alignment information for every read aligned by BWA in the Pediatric and Soil datasets; so, the following data have all been identified via alignment (we therefore do not consider reads that BWA did not classify). We then further denoted these classified reads as being identified by: BWA only and not by Kraken2 or WarpNL; BWA and Kraken2 but not WarpNL; BWA and WarpNL but not Kraken2; and by all three classifiers. Then, we parsed the alignment results, counted the number of individual base pairs aligned by BWA for each read pair, and plotted the density of these results for all classified sequences as well as on-target sequences (Figure 2.12, Figure 2.13).

The densities of aligned base pairs per sequence differed greatly in shape and location by algorithm. In the Pediatric dataset (Figure 2.12), the classifiers, when used together, identified query sequences where most of the base pairs in the read pair matched the database, which is evidenced by the high density around 200 bp (each read was approximately 105 bp in length, for a total of approximately 210 bp). The smaller density around 100 bp represents classifications of a single read within the read pair, likely where either the forward or reverse read matched to the database while its mate-pair did not. When all classifiers were used together, classifications favored reads that matched more exactly to the database, since reads classified by all algorithms satisfied a more stringent set of similarity criteria than reads classified by any classifier pair or any one classifier. This is similar to using multiple molecular diagnostic methods in combination to identify the presence of an organism, for example when testing in parallel with molecular diagnostic assays. Any result that is positive by the entire battery of tests is much more likely to be a true positive, if different methods are employed. Similarly, in a classifier ensemble, if reads are required to be positively classified by all classifiers in the ensemble, the resulting positive classifications tend to be very similar to the training database.

The densities for sequences that were classified by BWA and Kraken2 but not WarpNL show a higher density around the 100 bp mark with a relatively flat distribution elsewhere. This suggests that WarpNL does not as frequently classify read pairs where one read is similar to the database

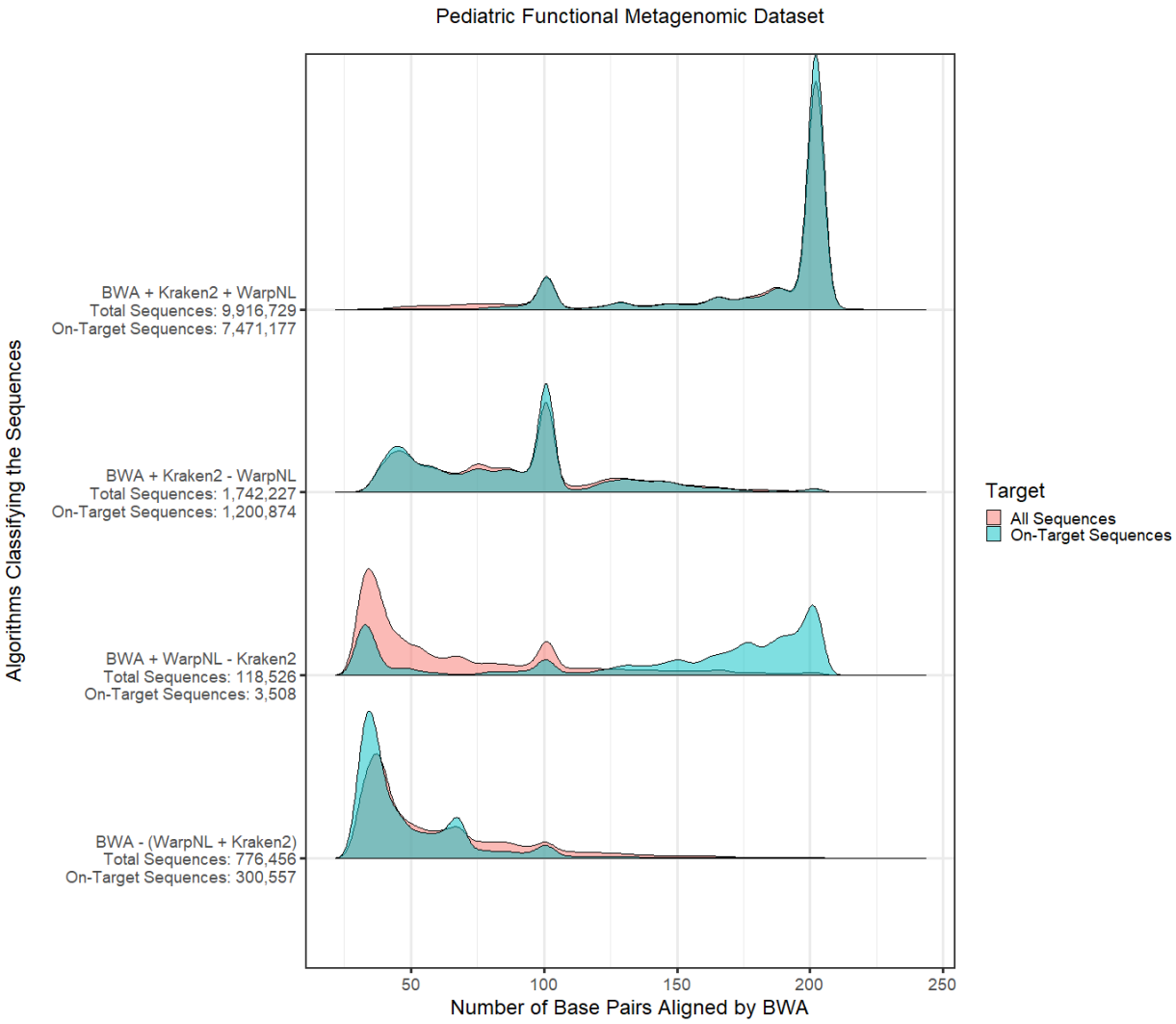

Figure 2.12: In the Pediatric functional metagenomic dataset, the density of the number of base pairs aligned per sequence differs according to which algorithm or combination of algorithms classified it. Sequences classified by all three algorithms had on average a higher number of aligned bases, with the peak densities at approximately 200 bp (both read pairs) and 100 bp (one read per mate-pair). This suggests that sequence read pairs classified by all algorithms are more similar to the MEGARes v2 database than those classified by individual algorithms or pairs of algorithms. Sequence read pairs classified by BWA + Kraken but not WarpNL show a peak density around 100 bp, suggesting that WarpNL is not good at classifying read pairs where one pair is genetically similar to the training database but the mate-pair is not. Read pairs classified by BWA and WarpNL both have high densities around small aligned base pair values (less than 50), suggesting they are capable of identifying small subregions of sequences that match to the training database where Kraken2 cannot. Note that the data used in this figure are produced by BWA (local alignment algorithm), so those sequences that were only identified by Kraken2 and/or WarpNL are not included.

while the other read is not. From an algorithmic perspective, this is not surprising, since WarpNL weights all *k*-mers equally during classification and examines all *k*-mers present in the read pair. In doing so, the sequence in the read pair that is not genetically similar to the database can often overwhelm its mate pair sequence that should match, causing a false negative classification. In comparison, Kraken2 and BWA are able to consider localized context (unequal weighting of *k*-mers or subsequences), since their algorithms are heuristic. Often, it only takes one exactly matching *k*-mer for Kraken2 to classify a sequence correctly, and BWA can likewise exclude extraneous information using local alignment strategies. This is an advantage of heuristic algorithms over machine learning classifiers that require use and equal weighting of the entire feature space, as is the case for the implementation of naive Bayes used in WarpNL.

Patterns in the reads classified by BWA and WarpNL but not Kraken2 demonstrate the point made earlier: that small *k*-mer classifiers and alignment methods capable of considering small sequence regions will identify sequences with fewer base pairs aligned than Kraken2 is capable of identifying. This is because Kraken2 uses a very large *k*-mer size (> 30), so if no exactly matching large *k*-mer is present in a read pair, it is invisible to Kraken2. Notice that the density towards the lower end of the aligned base pair distribution is centered roughly at 30 bp, which is roughly the *k*-mer encoding size of Kraken2. In this particular algorithm combination (BWA + WarpNL – Kraken2), there was also an apparent difference between the on-target and all sequence distributions, however this is likely due to the small number of reads present in this category (3,508). We expect that the on-target distribution would approach the shape of the all sequence distribution given enough reads classified in this category. However, this pattern could also be caused by a true difference in the amount of mutation away from the database present in these phenotypically resistant metagenomic products.

Finally, BWA without Kraken2 or WarpNL was clearly capable of identifying short subsequences that were missed by the *k*-mer classifiers. While it might be considered that these are false positive classifications, the sheer number (776,456 total and 300,557 on-target) of sequences classified and classified on-target suggest that at least a portion of them are real true positives. For

such sequences to be identified only by BWA, they likely have unmapped mate pairs (as previously discussed) and a large number of mutations present in the sequence. As previously shown in Figure 2.6 and Figure 2.7, BWA does have a region of sensitive detection at a high rate of mutation where it outperforms Kraken2 and WarpNL, likely due to the strategies implemented in the BWA-MEM re-seeding algorithm.

The alignment distributions show roughly the same patterns in the Soil dataset as in the Pediatric dataset (Figure 2.13). Additionally, as we previously hypothesized, there is a larger density of reads classified with a smaller number of base pairs aligned in the Soil dataset compared to the Pediatric dataset. This result suggests that the cloud of genetic variation in the Soil dataset less closely matches the genetic variation present in the MEGARes v2 database, causing fewer base pairs to be aligned per classified sequence on average. In these scenarios, the *k*-mer classifiers often have an advantage over alignment, since they are able to consider non-sequential and complex classification boundaries using long-range *k*-mer information that traditional seed-and-extend alignment strategies cannot replicate. We previously discussed that Kraken2 when used with WarpNL identified a large subset of sequences in the Soil dataset where BWA did not. However, such results wouldn't be reflected in Figure 2.12 and Figure 2.13, since the *k*-mer classifiers do not produce base pair-level information during classification and therefore were not able to be included in this analysis.

These results taken together emphasize that every algorithm has a different strategy and therefore a different behavior, even if they share the same training database. Instead of this being concerning, it should be leveraged to the analyst's advantage through the use of classifier ensembles, as demonstrated here. Since each algorithm provides a somewhat overlapping but unique view of the query dataset, they can be used in combination to achieve various tasks. For example, if the analyst wants to identify sequences that match to a database with extremely high confidence, they could require all algorithms to report positive classifications and exclude results that do not meet this criterion. Alternatively, if the goal was to identify potentially novel mutations away from known sequences, the analyst could require a positive classification by only one method and per-

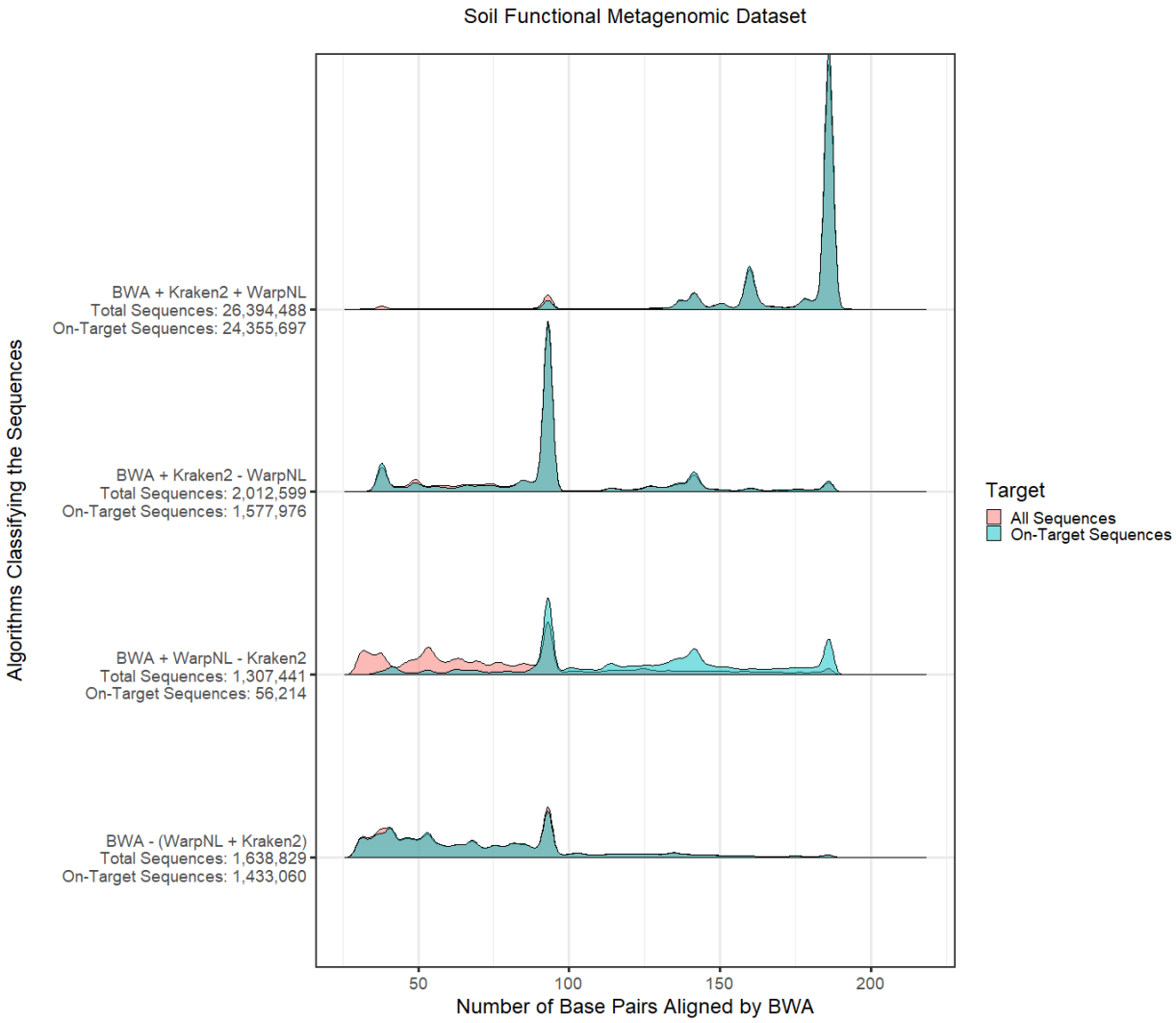

Figure 2.13: In the Soil functional metagenomic dataset, the density of the number of base pairs aligned per sequence differs according to which algorithm or combination of algorithms classified it. The density patterns shown here follow similar trends as in the Pediatric dataset. Additionally, there are an increased number of classified sequences with fewer base pairs aligned, which suggests that there is an increase in genetic distance away from the database overall in the Soil dataset as compared to the Pediatric dataset.

form further investigation using molecular approaches or other bioinformatics strategies like *de novo* assembly to validate putative positive classifications. At times, a single classification strategy may be suitable to a given task, however as computational resources continue to advance, it may be advised for analysts performing genetic sequence classification to consider two or more algorithms to improve accuracy.

Finally, we will briefly consider future directions of algorithmic development. The results presented here showcase classifiers for short-read sequence data, however modern sequencing strategies have begun to produce long-read sequences that will become an increasingly large proportion of query data for genetic sequence classification. It should be expected that the algorithms listed here will not perform as well on long-read query data as they do on short-read data. Though the problems may seem similar, many nuances are introduced into the classification problem as read length increases, i.e. one must consider localized regions (envelopes) of genetic sequences that can be different from neighboring localized regions. Therefore, the algorithms must be able to accommodate assignment of start and stop positions within a single query sequence, which none of the algorithms presented here can do in a single pass. This has been acknowledged by the developers of these classifiers as far back as the BWA-MEM publication [64]. However, many classifiers for long-read data have already been developed, including leveraging of old methods like hidden Markov models (HMMs), which have been a cornerstone of sequence classification for decades [73, 74].

Overall, the information present in the query sequence should increase drastically as read length increases, making classification an easier task in terms of identification but a more difficult task in terms of implementation. In the last several years, neural network machine learning strategies have evolved substantially in the field of natural language processing, introducing techniques like transformers and context approaches [75, 76]. We anticipate that such methods will be applicable to long-read data, particularly as database size increases to provide these more complex models with the information needed to identify deep contextual patterns in the training data. Hopefully, as the amount of information increases with read length, these more advanced classifier methods

that consider sequential context can help provide an increasingly robust picture of the structural organization of genetic sequence data.

Error correcting strategies can improve classifier sensitivity

Much of the previous sections on classification algorithms and genetic sequence databases involved discussion of methods to improve the sensitivity of sequence classification without sacrificing accuracy through loss of specificity. We have previously discussed that the most advantageous way to capture more genetic variation during the classification task is to improve the robustness of the underlying training database, and, to a lesser extent, through the use of algorithm ensembles and more representative feature encoding strategies. However, other methods for capturing increased genetic variation have also been explored with variable success, largely borrowing ideas from the field of signal processing [77]. An important aspect of the signal processing field is to encode and sample information in an efficient but redundant manner, such that if pieces of information are lost during transmission of a signal from a sender to a receiver, the message can be reconstructed intact by the receiver despite the loss of information. An example of this is the encoding of data packets such that packet loss during internet or cellular data transmission does not result in substantial information loss on the receiving end [78].

This concept can also be leveraged to some degree in genetic sequence classification by thinking of mutations with respect to the database as "errors" (i.e. as lost or corrupted information). With *k*-mer classifiers in particular (or seeds for seed-and-extend aligners), misclassifications occur when the *k*-mer does not match the database, or if a mutated *k*-mer causes a mismatch to a different database class. If such a mismatch is caused by only a few mutations in a single window of length k base pairs, the *k*-mer can be correctly matched if those mutations can be ignored. This is analogous to correcting "errors" in the mutated sequence such that it matches correctly to the database. However, if too many errors are "corrected," it is also possible to increase the false positive rate by being too permissive during classification. Therefore, methods that seek to gain sensitivity through error correction during feature encoding must balance an increase in sensitiv-

ity with a decrease in specificity of classification, typically through the careful selection of the error-correcting strategy.

The Kraken family of classifiers introduced this idea into their algorithm by using "spaced seeds"; this was originally introduced into Seed-Kraken but was also included in the implementation of Kraken2 [57]. Using a specifically chosen, fixed set of gaps during *k*-mer feature encoding (determined through experimentation), Seed-Kraken and Kraken2 were able to improve the sensitivity of classification by roughly 2-5%. The inspiration for this method was taken from early experimentation of spaced seeds from seed-and-extend alignment methods [79], hence the use of "seed" in the name. However, the approach of error correction goes back further into the fields of information theory and signal processing.

A more dated error correction strategy comes from low-density parity-check (LDPC) codes, first introduced by Robert Gallager in 1962 [80]. Instead of using a single gapped sequence to encode *k*-mers, LDPC codes (applied to *k*-mers) evenly subsample a *k*-mer into *l*-mer subsequences, where $l < k$ and l divides k without remainder. Gaps are introduced into the *l*-mers, creating a kind of bootstrap-like subsampling pattern that evenly covers the *k*-mer space; this strategy is traditionally represented in a parity check matrix (Figure 14). In recent work, the metagenomic binning software Opal leveraged LDPC codes to some degree to improve the sensitivity of genetic sequence clustering (aka metagenomic binning), an unsupervised clustering task with strong parallels to supervised genetic sequence classification [81].

In the WarpNL classifier, we also implemented the Gallager LDPC strategy to see if it improved the sensitivity of supervised classification using small *k*-mers and the naive Bayes machine learning classification algorithm. We utilized k=30 and l=10 with a bootstrap row-weight of 2 (i.e. every base pair in every *k*-mer is sampled exactly 2 times during feature encoding) (Figure 2.14b). In Figure 2.15, we show via receiver operating characteristic (ROC) curves that this resulted in an overall decrease in sensitivity and specificity on the previously introduced simulated MEGARes v2 data through a total reduction in area under the curve (ROC-AUC), a method used to visually evaluate classifier accuracy. Additionally, we evaluated the performance of the Gallager encoding

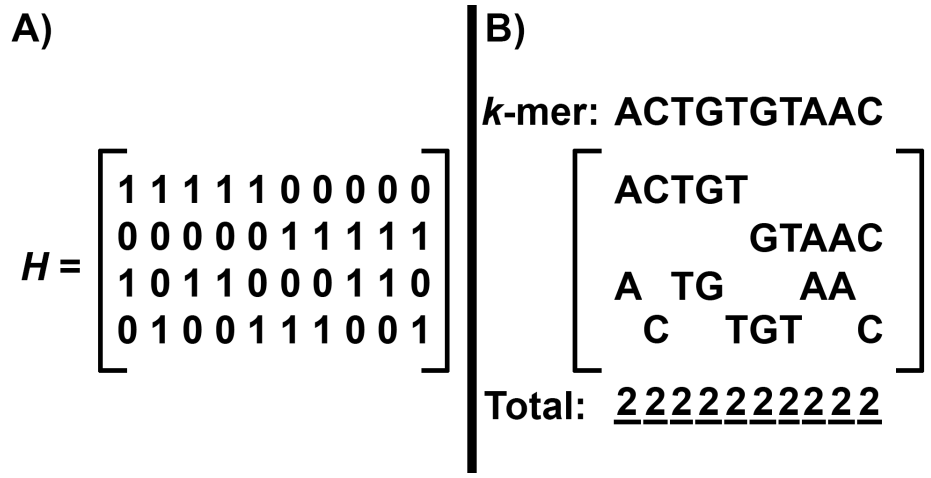

Figure 2.14: Gallager LDPC error-correcting codes can be used over *k*-mer features to increase classification sensitivity. A) The parity check matrix identifies which positions are encoded for any given *k*-mer, indicated by the non-zero bits. The column sums indicate the row weight of the LDPC design. B) The LDPC matrix on the left is shown for an example *k*-mer sequence

strategy on the previously introduced Pediatric and Soil functional metagenomic datasets (Figure 2.16, Figure 2.17). The classification patterns resulting from WarpNL with Gallager encoding (WarpNL-Gallager) consistently agreed with the WarpNL classification patterns using standard *k*-mer encoding (WarpNL-Multinomial) across all samples from both the Pediatric and Soil datasets. As seen in the Venn-diagrams, there were very few sequences for which WarpNL-Gallager and WarpNL-Multinomial disagreed, and both *k*-mer encoding strategies matched results produced by Kraken2 and BWA in a similar manner. If the WarpNL-Gallager method truly improved on sensitivity of detection over WarpNL-Multinomial, we would expect the ROC-AUC to be greater for WarpNL-Gallager (Figure 2.15) and the WarpNL-Gallager method to diverge more dramatically from the classification patterns produced by WarpNL-Multinomial in the Venn-diagrams (Figure 2.16, Figure 2.17). Therefore, while spaced seeds and LDPC codes can be used to some effect, as shown by Kraken2 and Opal, they did not appear to increase sensitivity when used with the WarpNL classifier. Regardless, we deemed it pertinent to include this discussion of spaced seeds in this work, since they can help to improve the sensitivity *k*-mer classification in certain contexts.

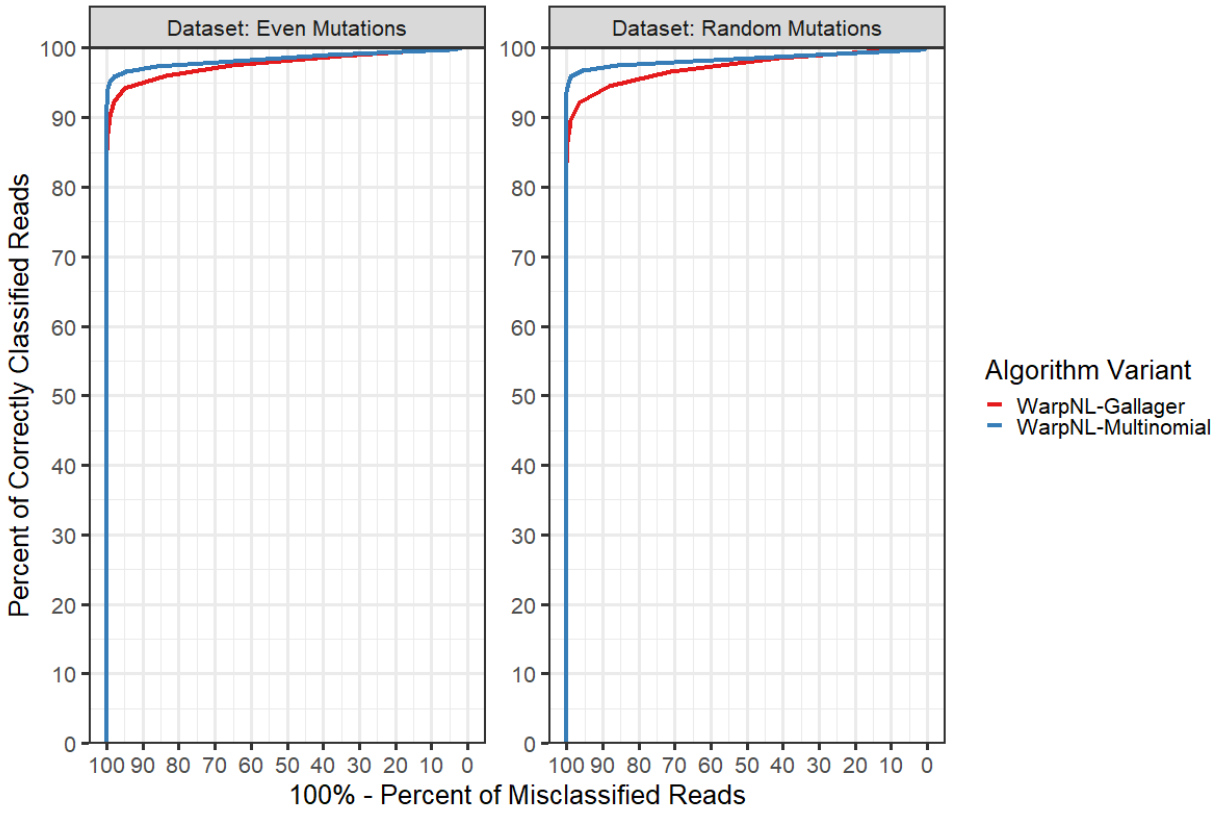

Figure 2.15: Gallager LDPC encoding used in the WarpNL classifier (WarpNL-Gallager) showed an overall decrease in ROC-AUC for both the simulated even mutations and random mutations datasets. We therefore concluded that Gallager LDPC encoding did not improve sensitivity of sequence classification in the face of either kind of mutation pattern, so far as it was implemented in the WarpNL software. Here, traditional sensitivity is shown on the y-axis, and 1 – specificity is shown on the x-axis, which is typical of receiver operating characteristic (ROC) curve diagrams. The area under the curve (AUC) can be calculated by taking the area under this ROC curve.

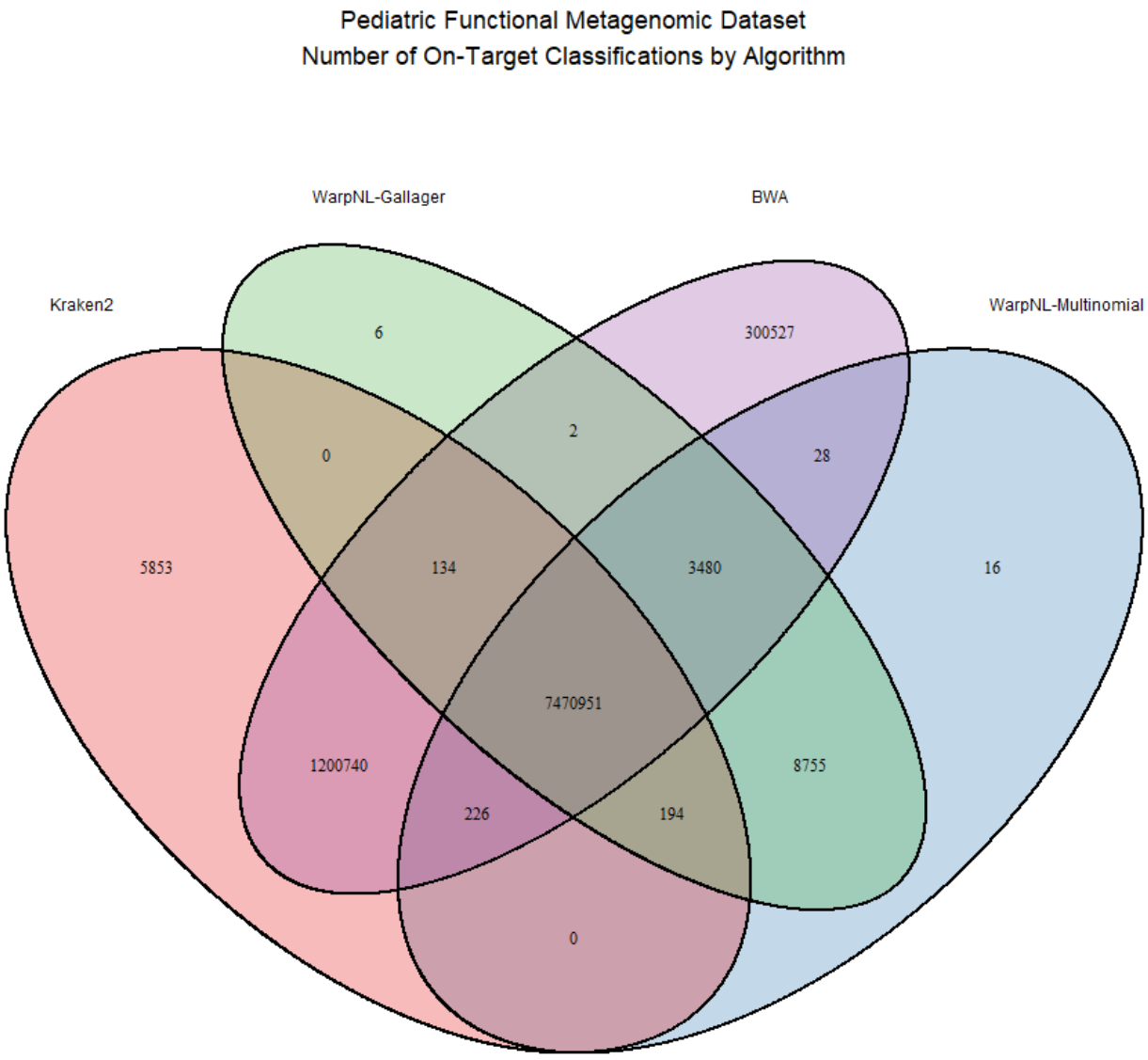

Figure 2.16: On the Pediatric functional metagenomic dataset, the standard *k*-mer encoding (WarpNL-Multinomial) and Gallager LDPC encoding (WarpNL-Gallager) demonstrated largely overlapping classification patterns when compared to the other classification algorithms.

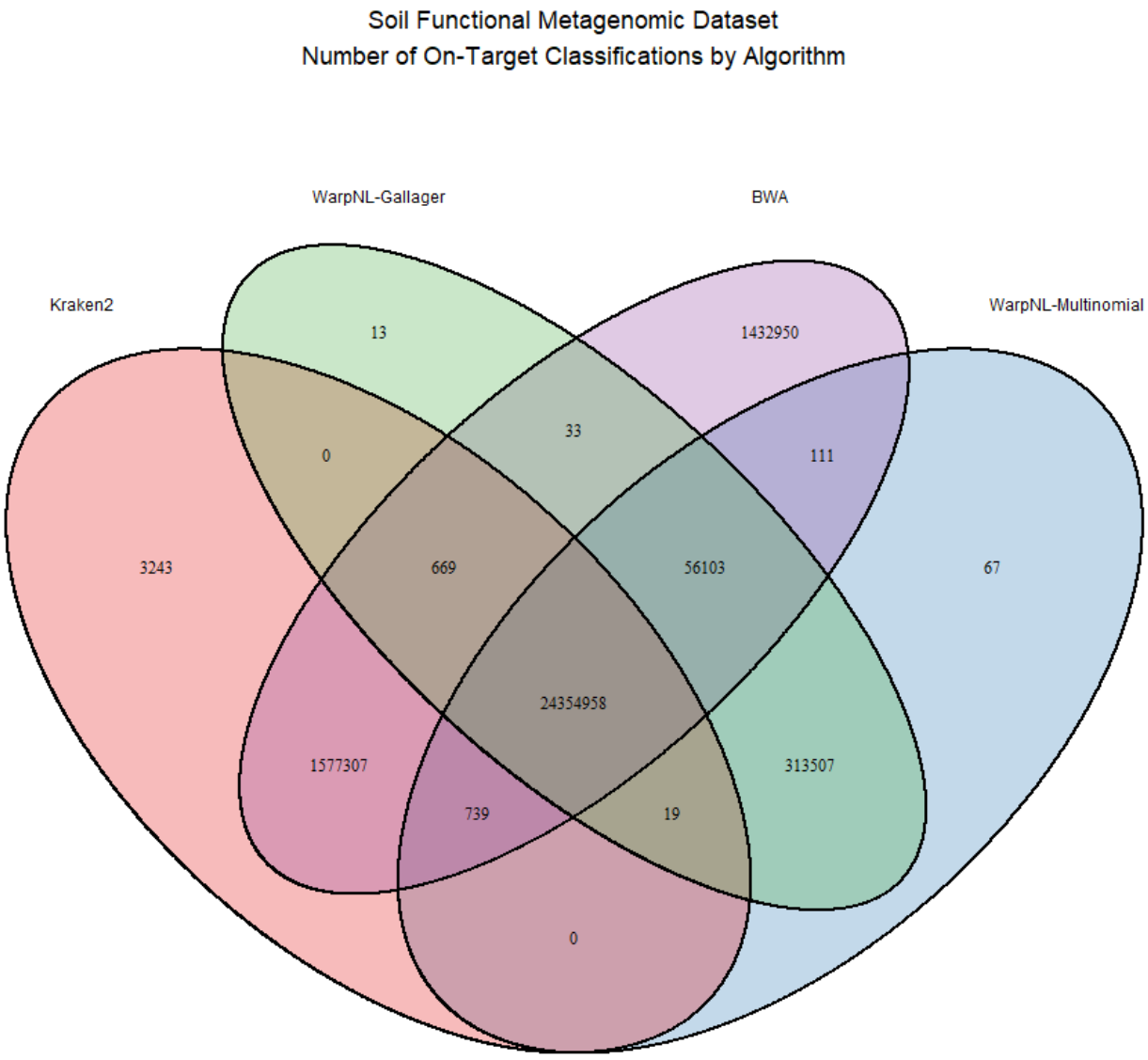

Figure 2.17: On the Soil functional metagenomic dataset, the standard k-mer encoding (WarpNL-Multinomial) and Gallager LDPC encoding (WarpNL-Gallager) demonstrated largely the same classification patterns when compared to the other classification algorithms.

Strategies for determining which sequences are classified versus unclassified are critical to classifier performance

Much attention is given in practice and in the literature to the task of assigning class labels for supervised genetic sequence classification, i.e. the focus is often on differentiating *Escherichia coli* from *Salmonella enterica* from *Saccharomyces cerevisiae*. However, genetic sequence classification also involves the step of determining whether or not a query sequence is genetically similar enough to the database to be classified at all. Here, we refer to this as the "unclassified problem," and we argue that determining whether a sequence should be classified or not is often the more difficult of the classification tasks. Despite the importance of this step, it is not commonly discussed in detail in classifier literature.

The unclassified problem is a difficult subject, because it either forces the developer of the classifier to use an arbitrary threshold for determination of unclassified labels (for heuristic methods), or it requires statistical cross-validation to either tune this threshold or train an additional learned model to identify unclassified decision boundaries. In the latter case, learned parameters and cross-validation procedures must be tuned on a case-by-case basis that changes with the training set; thus, a decision boundary learned by a model using one database will not transfer to other databases without retraining of the model on the new database. In many cases, cross-validation and tuning also require the user to provide "outgroup" sequences that are sufficiently different but not too different from the database, such that appropriate decision thresholds and boundaries can be learned. This severely hampers end-user usability, particularly if the classifier is aimed at reaching a wide audience of non-bioinformaticians or end-users that are not trained in machine learning procedures. The ideal classifier, from an end-user perspective, would function equally well with all databases and not require tuning to the specific classification task, as algorithms like BWA and Kraken2 aim to achieve. However, this end-user ease of use is often impossible to achieve when using more complex classification methods, such as most machine learning models. This includes WarpNL, which requires the use of both an outgroup training set as well as statistical cross-validation to train the outgroup classifier model.

The unclassified problem is a theory-rich subject that drives at deep structural patterns in biological sequence data, which is why it is often difficult to get right in practice and often circumvented during the creation of classifier algorithms. We will expand on these assertions in this section and then discuss the pros and cons of how WarpNL handles this problem. From a mathematical perspective, it is helpful to understand that biological sequence data are already highly structured. This structure is a result of the multi-factorial, complex interactions that arise from a biological system evolving under selective pressure. When we seek to classify genetic sequences at the DNA level, we are performing a classification task that is several steps removed from the physical organisms and proteins that are under forces of natural selection. While small changes in DNA look like small differences to the sequence classifiers, they can actually result in drastic phenotypic effects, including organism nonviability. For example, a SNP at the first nucleotide position in a translational reading frame codon that encodes a highly conserved protein could have large phenotypic impacts that the classifier cannot see at the DNA level.

However, the sequences in the training databases are a result of this complex evolutionary system and are therefore highly organized. Thus, when we seek to choose sequences with which to train an outgroup versus ingroup classifier (or to learn a threshold parameter for heuristic algorithms), the outgroup sequences must be genetically close enough to the highly structured database sequences to properly learn the parameter. These two datasets must be related to one another enough for the decision boundary to be refined correctly but also distant enough that the algorithm doesn't develop a decision boundary that is too strict to capture genetic mutations in future query data. If the distance between the ingroup data (training database) and the outgroup data is too large, then the unclassified decision boundary can be chosen from a wide range of solutions/locations and still separate these two distant datasets. However, when applied to real query data, this threshold is nonsensical, because other biological data that is under selective pressure is non-random and will be much more genetically similar to the database than the distant outgroup used to learn the decision boundary. Likewise, if the outgroup data are too similar to the ingroup data, the unclassified

decision boundary may be optimized to be inappropriately close to the ingroup data, which will increase false negative rates of downstream classification.

The ideal placement for the unclassified decision boundary is in a location that excludes sequences with a high enough mutation rate such that the unclassified sequences are not phenotypically related to the training database. Said differently, the sequences that are unclassified should ideally be biologically interpretable as being unrelated to the database sequences by the analyst. Based on the data classified in the simulated and functional metagenomic datasets presented here, we determined that an upper bound for a reasonable decision boundary for the MEGARes database was around a 10% mutation rate, which equates to roughly 15 mutated bases per 150 bp sequence read (or a smaller number of mutations for subsequence regions classified by alignment). However, the desired mutation rate's upper bound will vary depending on the biological system under consideration, since mutation rates differ by organism class and biological niche [82].

For classifiers that use a heuristic threshold to make unclassified determinations, the algorithm can simply use the genetic sequence and the number of mutations or gaps encountered to make this determination (i.e. the percent identity of alignment). However, for *k*-mer classifiers, it is more difficult, since a single mutation in a 150 bp sequence translates into k counts of non-matching *k*-mers, assuming that one uses overlapping *k*-mers with a single base pair shift. The shorter the read or the more mutations there are, the higher the frequency of mismatching *k*-mers. This is why with short-read data, Kraken2 often considers a single matching *k*-mer as being a candidate for ingroup classification, since a few well-spaced mutations and large k size means that the majority of *k*-mers in a 150 bp sequence will mismatch against the database. Therefore, Kraken2 can simply count *k*-mers and perform ingroup classification on sequences with any matching *k*-mers, which performs well enough to sidestep the unclassified problem entirely.

However, for small *k*-mer classifiers, a higher frequency of query *k*-mers will match to the training database up to the desired 10% mutation rate threshold if k is less than approximately 15, depending on sequence read length. For WarpNL, we leveraged this fact to create a support vector machine classifier (SVC) that performs outgroup versus ingroup classification prior to the naive

Bayes classification step [83]. To keep the outgroup classification step computationally efficient, we used a linear decision boundary and only two dimensions of data for each sequence: the percent of forward read *k*-mers matching the database on the x-axis, and the percent of reverse read *k*-mers matching the database on the y-axis (Figure 2.18). If single-end data (only forward reads) were provided, then the mismatch percent calculated for the forward reads (x-axis) was also used as the mismatch percent for the non-existent reverse read (y-axis). The data seen in Figure 2.18 was used to learn the WarpNL SVC unclassified decision boundary. These data were produced by calculating the *k*-mer mismatch percentages for a combination of "left out" sequences from K-fold cross validation and a set of user-provided outgroup sequences (see methods for details).

Ideally, these data would separate into two well-defined and mostly non-overlapping clusters such that a tightly constrained but accurate decision boundary could be learned between the in-group and outgroup data. However, note that the data left out by K-fold cross validation (with ingroup labels, colored in red, Figure 2.18a) occupy space in both the low *k*-mer mismatch area of the graph (bottom left coordinates) and the high *k*-mer mismatch area (top right coordinates). This is because when data are left out from the database ontological categories during the K-fold cross validation process, sometimes the sequences that are left out no longer have a representative sequence member remaining in the subsampled training set. This harkens back to the point made earlier in Section 2.4.2: that sequences with the same ontological label can be more genetically distant from each other than one might expect, either due to the way the database curator designed the database ontology or due to natural variation between genetic sequence clusters that share the same ontological category. For example, if we were classifying bacterial sequences at the Species taxonomic level, there might be sequence regions that differ substantially between E. coli K12 and E. coli O157:H7 strains; if we, by chance, left out all K12 strains during a cross-validation fold, the sequence regions unique to K12 strains would no longer have a representative reference genome in the training set, causing reads produced over such regions to have high *k*-mer mismatch values. This is why we see that most of the ingroup (red) labels have a low percentage of *k*-mer mismatch values but a smaller subset of left out sequences have higher mismatch values.

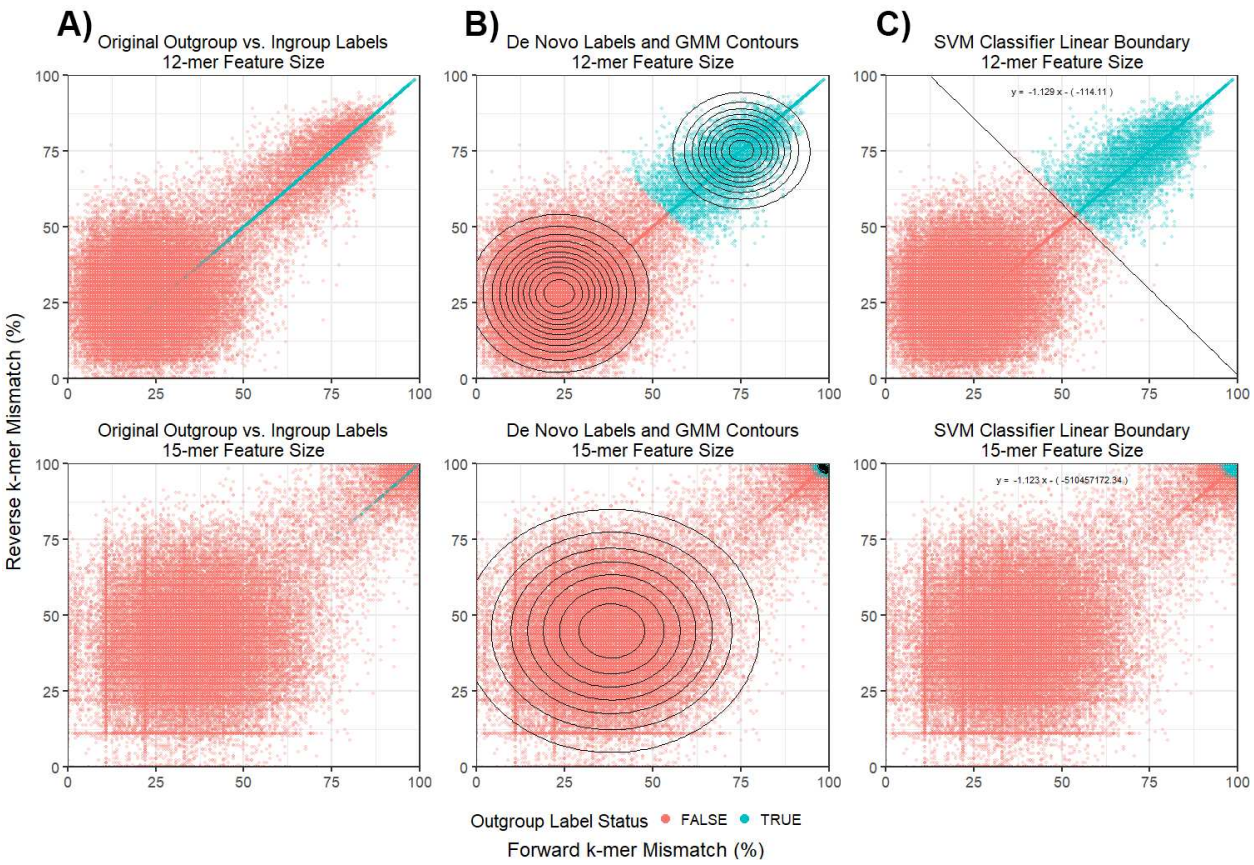

Figure 2.18: WarpNL learns to classify outgroup (unclassified) versus ingroup (classified) sequences through a two-step learning process using a Gaussian mixture model (GMM) followed by a linear support vector machine classifier (SVM/SVC). In the rows, the steps of this process are displayed for k=12 (top) and k=15 (bottom). A) The red labels are the ingroup sequences that were "left out" during K-fold cross validation; these sequences theoretically should be classified back to the database as ingroup sequences. The blue labels are user-provided outgroup sequences (here, they occupy the off-diagonal because they are single-end reads, thus the x-axis is the same as the y-axis for these sequences). B) A GMM re-labels the distant sequences as outgroup sequences such that the SVC boundary can be learned properly. C) The linear SVC boundary is then learned via optimization of the SVM dual problem. Note that the GMM and SVC optimizations work well when k=12 but break down for higher values of k, such as k=15.

Another way to think about these left out sequences with high mismatch values is that they no longer truly belong to the ingroup label and should be re-labelled as outgroup (blue) for the purpose of learning the unclassified SVC decision boundary. To perform the task of re-labelling in the most unbiased way possible, we used a bivariate Gaussian mixture model (GMM) unsupervised clustering algorithm, which produced the *de novo* labels and contours seen in Figure 2.18b. Then the linear SVC decision boundary was learned using the entire dataset across all K-folds and the user-provided outgroup sequences, the result of which is shown in Figure 2.18c. In general, this method worked effectively for smaller values of k, including the optimal value of k=12 that was used for all of the WarpNL outgroup classifications in this work (top row, Figure 2.18).

A disadvantage to the above approach is apparent when the value of k increases beyond this point, for example when k=15 (bottom row, Figure 2.18). At larger values of k, having even a small number of mutations away from the database results in high *k*-mer mismatch percentages, which forces the outgroup cluster to occupy a small area of space in the top right corner of the graph. This makes fitting the GMM and SVC difficult and can produce nonsensical decision boundaries, as seen in the boundary equation displayed on the bottom right panel in Figure 2.18. Therefore, while the above method for fitting the unclassified boundary worked well for WarpNL with k=12, it would need to be adapted individually for every unique combination of training database, k value, and user-provided outgroup sequence dataset.

The pattern shown in the bottom row of Figure 2.18, where a small number of mutations translates into large mismatch percentages as k increases, further illustrates the previous point that Kraken2 can circumvent the unclassified problem entirely with large enough k. k=15 is less than half of the value Kraken2 uses for its *k*-mer encoding, and k=15 already produces high mismatch values with few mutations away from the training database. As k becomes even larger, the analyst can simply assume that any mutation away from the database belongs to an outgroup *k*-mer, taking the pattern seen in Figure 2.18 toward its limit. Likewise, alignment simply finds regions of local alignment and leaves the decision of whether such aligned subsequences truly belong to the database up to the analyst, thereby also circumventing the unclassified problem. The strategies

used by both BWA and Kraken2 are therefore arbitrary and founded only indirectly on confidence metrics or statistical theory, which we consider a disadvantage to their approaches.

Hopefully, as the field continues to progress, there will be more focus on the integration of statistical measures into computer science algorithms and heuristic techniques. This would leverage the advantages of the partial alignment and alignment approaches while providing the user with a measure of confidence that they could use to make determinations against the unclassified problem. At the time of this writing, we notice that such integrations of statistical methods (particularly Bayesian approaches) with computer science algorithms is already taking place in genetic sequence classification, such as with the UNCALLED classifier for long-read Nanopore data [84]. We anticipate that such methods will become commonplace in the future, and we suspect new approaches will reveal further information about the rich problem of mathematical structure in biological sequence data, both for the unclassified problem and otherwise.

2.4.5 Classifiers are constrained by availability of computational resources

Genetic classification is constrained by the availability of computational resources in two ways: 1) the choice of classification algorithm, feature encoding strategy, and database may change depending on the computer hardware available to the analyst, and 2) advances in computational capabilities at the cutting-edge have enabled the use of methods that were previously not feasible, as is the case here for WarpNL. The first point is important for the bioinformatics community to keep in mind, since not all analysts and researchers have access to distributed supercomputer clusters; many analysts are performing sequence analysis on modest hardware like laptops. As a result, it is important to have a variety of genetic sequence classification strategies available in the field that can meet the needs of both the laptop and supercomputer cluster compute niches. Tools like BWA and Kraken2 help fill this role due to their low computational overhead and forgiving end-user experience. The second point is also interesting: many of the mathematical techniques that are now commonplace, like neural networks, were only widely adopted due to recent advances in computer technology. As computer platforms continue to advance and applications like graphics

processing units (GPUs) become more commonplace in bioinformatics, it may be worth revisiting older but more costly techniques, as we have done here with naive Bayes. To emphasize how much more costly the naive Bayes algorithm is, we provide a runtime comparison of WarpNL to BWA and Kraken2 below.

We classified an increasing number of simulated sequences against the MEGARes v2 database using BWA, Kraken2, and WarpNL to demonstrate their relative computational time complexities (Figure 2.19). Wall clock time (the time experienced by the user, aka real time) was recorded for each algorithm as the number of simulated reads scaled from several thousand to over 25 million. Twenty CPU threads were used for BWA and Kraken2, and 4 GPUs were used for WarpNL. Given the small size of the MEGARes database, BWA performed the fastest with a linearly-scaling trend in time as read number increased. Kraken2 also scaled linearly with very fast compute times, classifying all reads in under 40 seconds, since each *k*-mer lookup in the query data takes only $O(1)$ due to its efficient data structures. WarpNL demonstrated two roughly linear phases of time scaling but took much longer to compute on the data than Kraken2 and BWA. This is due to its time complexity being $O(m4^k)$, where m is the number of query sequences and k is the *k*-mer size. While the scaling of m is linear (as shown in the graph), the coefficient against which it is multiplied is very large, resulting in much longer compute times than the other two algorithms. These results illustrate the differences in algorithm design and behavior between the different classifiers and emphasize the need for the analyst to choose a tool that fits the classification task but also conforms to the computational resources available.

2.5 Conclusion

The results presented in this work tell a comprehensive story of how supervised genetic sequence classification is performed in practice and how a selection of classifier algorithms behave on concrete data. There are five major aspects to genetic sequence classification, all of which must be considered as integral to the classification task: the nature of the query data, the robustness of the reference database, the method of feature encoding, the choice of classifier algorithm, and the

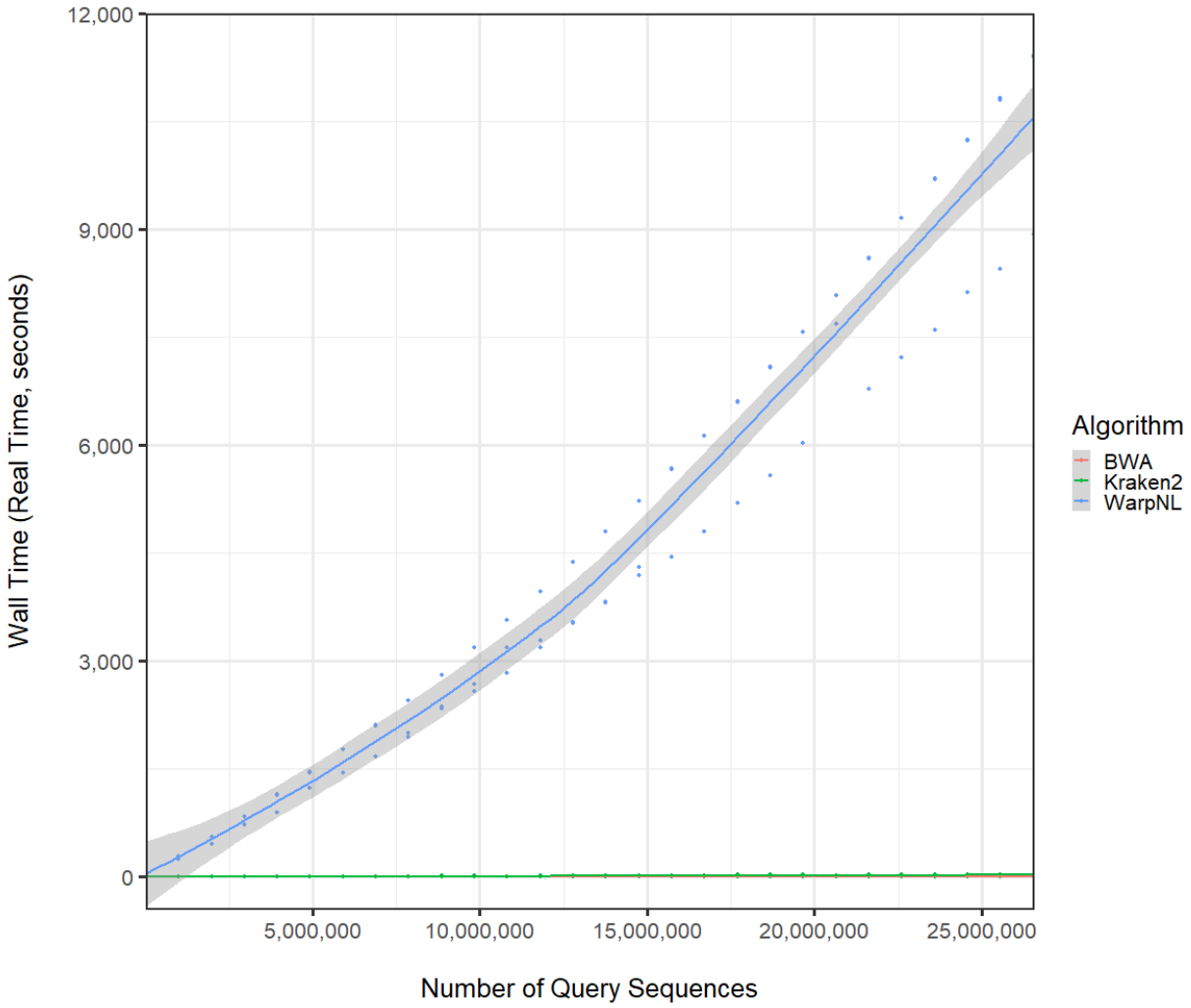

Figure 2.19: Kraken2 and BWA have a low time complexity, while WarpNL has a higher time complexity, as evidenced by the trends wall clock time as the number of query sequences increases. While all three algorithms scale roughly linearly in this graph, WarpNL has a higher coefficient of scaling (4^k) since it considers all possible k-mers during the classification process. While the WarpNL curve may appear exponential, it is roughly linear in two segments, with the curvature being caused by additional CPU and operating system operations for memory management of large data structures. A truly exponential algorithm would scale to large values more rapidly than this.

availability of computational resources. While much of the focus in the literature tends to be on improvements to classification algorithms, these other four areas are also of critical importance to the success of the classification task and deserve an equal amount of attention, particularly novel methods of feature encoding and the development of robust databases.

We also acknowledge several limitations of the work presented here: each of the major subjects addressed here could be their own focus of study, and it is therefore outside the scope of what we can fit in this work to consider nuances in each of these areas. In particular, more could be said about feature encoding strategies and the intricacies of different methods. Additionally, we used only one reference database in this work (MEGARes v2), however additional databases could be explored using similar simulated and gold standard data strategies. We chose the MEGARes database due to its moderate size and well-organized ontological structure, which allowed us to showcase the naive Bayes algorithm without resorting to advanced strategies for parallel computing, such as the use of distributed compute clusters. Further work could be done using a variety of databases in different areas, including databases focusing on viral, eukaryotic, or microbiome sequences. Finally, we did not consider more than three major algorithmic classes, since there are so many available in the literature: we chose to focus on popular methods that have withstood the test of time and are still in widespread use, since most readers would be familiar with these tools. We also focused on short read data, however similar discussions and arguments could be made for long read data [85].

Going forward, we anticipate that reference sequence databases will continue to advance in robustness, size, and variety of ontological structure, since several efforts already exist toward this aim [46, 86, 87]. We also anticipate that long read sequencing technologies will become more popular, which will necessitate the creation of new strategies for long read sequence classification. Such strategies will need to consider localized subsequences as separate classifiable units, similar to how Markovian classifiers like HMMs already function [73]. Overall, we hope to have provided a thorough introduction to genetic sequence classification for those new to the subject while pro-

viding enough detail in this work to help experienced bioinformatics analysts gain insight into the practical behavior of algorithms used in their daily work.

2.6 Methods

2.6.1 Condensed WarpNL Classifier Methods

The WarpNL classifier software takes data in FASTQ format as input (paired- or single-end) and outputs classifications of each FASTQ read (or read pair) to the user-provided database of FASTA-formatted sequences. The overall approach of WarpNL is to encode genomic data into a binary string of fixed length and use these binary strings as features to perform naive Bayes classification. In the subsections below, we provide a brief summary of methods for each step in the WarpNL classifier. Extended methods related to the WarpNL software can be found in Chapter 3.

WarpNL sequence encoding methods

WarpNL encodes standard nucleotides (A, C, T, G) into two-bit binary representation. Ambiguous nucleotides are randomly chosen to be one of the standard nucleotides, and the RNA nucleotide uracil is translated into its DNA equivalent thymine. WarpNL concatenates these binary representations into a fixed-length binary string, which represents the "word size" or *k*-mer, a genomic feature of length k nucleotides. Therefore, the resulting binary string is 2k bits long and represents an integer in the range of $[0, 4^k)$. These features are then used as an index to increment an integer count vector of length 4^k, effectively counting the number of times a unique *k*-mer appears in a genetic sequence.

WarpNL uses two encoding schemes: multinomial and Gallager. In multinomial encoding, the nucleotides forming each *k*-mer are contiguous, and each sequence of length L has exactly $L-k+1$ overlapping *k*-mers. This encoding scheme is standard for *k*-mer classifiers and has been commonly used for naive Bayes classifiers elsewhere [4]. The Gallager encoding scheme uses both contiguous and gapped, non-contiguous *k*-mers according to the Gallager low-density parity

check encoding method commonly used in error-correcting codes in other computer science fields [80]. This method was introduced by the Opal sequence classifier [81] to increase classification sensitivity in the face of sequence mutation away from the reference database. See S1 File for mathematical details on these encoding schemes.

WarpNL reference database encoding

The user provides a database of reference sequences in FASTA format, which contains the genetic targets for the classifier. WarpNL also requires an annotation structure for the reference database that maps the reference sequences to annotation nodes occupying a directed graphical tree (non-cyclic, diverging annotation graph, like the taxonomic tree of life). Using the annotation tree, WarpNL builds a matrix of features represented in the reference database. Each column of the database matrix represents a leaf (terminal, lowest-level) node on the annotation graph, while each row represents a *k*-mer feature. The entries of the matrix are decimal probabilities, which are constructed by counting the features for each column (node), adding a Lidstone pseudocount of 1 to each *k*-mer feature, and dividing by the column sum, such that the columns sum to 1 according to a multinomial distribution [88]. These probabilities are then log-transformed to facilitate subsequent naive Bayes classification. WarpNL also stores which *k*-mers were observed across all sequences in the database to assist in subsequent steps.

WarpNL query sequence encoding

The user provides one (single-) or two (paired-end) sequence files in FASTQ format that contain the query sequences to be classified. If paired-end data are used, each read-pair is treated as a single observation. Query sequences are encoded in the same manner as described for the reference database, producing a query matrix where the rows represent each read or read-pair, and each column represents a *k*-mer feature. The query matrix entries are left as non-negative integer counts, as opposed to the reference database matrix, where they were normalized to sum to 1.

WarpNL sequence classification

Using the query matrix (Q) and the database matrix (D), reads or read-pairs can be classified by taking the inner product of QD, where Q is of dimension M rows by K columns, and D is of dimension K rows by N columns. The result matrix (R) is then of dimension M rows by N columns, where the rows represent each read or read pair, and the columns represent each leaf node in the reference database. The entries of the result matrix are the maximum *a posteriori* probability estimates of each read or read-pair against each leaf node in the reference database, on the log-scale. A classification can then be determined for each row by finding which entry contains the row maximum. This procedure arises from a standard derivation for naive Bayes classification, which is provided in Chapter 3.

WarpNL outgroup determination algorithm

A difficult task for modern sequence classifiers is determining which query sequences are too genetically distant to be classified against the reference database, resulting in an "unclassified" call. To make this determination, WarpNL tracks the percent of *k*-mers in the query sequence that are also present in the reference database for the sequence (single-end) or forward and reverse read-pair (paired-end). These percentages of forward and reverse *k*-mer matches are used as input to a two-dimensional, linear support vector classifier, which labels sequences as classifiable or unclassified. Sequences that pass the support vector classifier threshold (i.e. are genetically similar enough to the reference database to classify) continue to the WarpNL naive Bayes classification step described above. The support vector classifier is trained using both the reference database and a user-provided file of "outgroup" sequences that are genetically distant from the reference database. Standard k-fold cross validation was used to optimize the support vector classifier bias hyperparameter [83, 89].

WarpNL GPU acceleration and distributed processing

WarpNL requires CUDA-capable (NVIDIA architecture) graphical processing units (GPUs) to accelerate the naive Bayes classification process described above. The WarpNL software includes

an automated scheduler that determines capabilities of both CPU and GPU resources and schedules jobs optimally according to available resources. Acceleration is available for classifier training as well as parallel processing of query sequences.

2.6.2 MEGARes v2 antimicrobial resistance database modification

All classifiers examined in this work utilized the publicly available MEGARes antimicrobial resistance database (v2.0.0, 14 October 2019) [17]. Modifications to this version of the database were made to improve classifier performance. To determine which modifications were necessary, the WarpNL classifier in multinomial mode was used to classify the database against itself, which should result in perfect classification. Results that were misclassified to an inappropriate annotation were examined, and their sequences were queried against the NCBI database using the nucleotide and translated nucleotide BLAST algorithms (BLASTn and BLASTx) on the publicly available BLAST website. These misclassified entries were corrected based on agreement from BLAST and WarpNL classification results or were removed from the database if the BLAST classification did not match the WarpNL classification. The modified MEGARes database used in this work was included in the WarpNL source code repository (S1 Code).

2.6.3 Construction of the MEGARes v2 Kraken2 database

sing the modified MEGARes v2 database described above, the annotation structure was converted into Kraken2 format using custom Python scripts. The MEGARes vs2 Kraken2 database and scripts utilized in its construction were provided in the WarpNL publication code repository (S2 Code).

2.6.4 Methods for classifier speed and memory performance benchmarking

Comparisons for classifier speed and memory consumption were made by varying the number of query sequences using the entire modified MEGARes v2 database. Because of algorithmic differences, it was not meaningful to vary the number of reference database sequences or annotation graph composition across the classifiers: sequence alignment with BWA-MEM depends on the to-

tal number of reference database sequences, whereas Kraken2 and WarpNL depend on total *k*-mer content and number of annotation tree leaf nodes, respectively. Therefore, we focused on comparisons in classifier speed and simply provided average memory consumption across experiments as an additional metric.

To compare classifier speed, 150 bp reads were simulated from the modified MEGARes v2 database without mutation to produce datasets that vary in the number of query sequences. Each classifier (BWA-MEM, Kraken2, and WarpNL) were then run against these datasets in 5 replicates to capture average runtime. For BWA-MEM and Kraken2, 20 CPU threads were used to parallelize sequence processing, and results were reported for total runtime as well as runtime per thread. For WarpNL, 4 GPUs were used in parallel, and results were reported for total runtime as well as runtime per GPU. All benchmarks were performed on a workstation running an Ubuntu Linux 18.04 LTS operating system with an Intel Core i9-9980XE 18-core 3.0 GHz CPU, 128GB DDR4 2666MHz DRAM, and 1TB PCIe Gen 3.0 M.2 2280 solid state drive, including 2x NVIDIA RTX 2080Ti 11Gb GDDR6 and 2x NVIDIA GTX 1080Ti 11GB GDDR5 GPUs connected individually via PCIe 3.0 x16 slots (without SLI or NVLink connections).

2.6.5 Methods for simulated read experiments

The data used in the simulated read experiments were generated using custom Python scripts. Two datasets were generated using the modified MEGARes v2 database: even mutations and random mutations. For both datasets, single-end short reads of 150 bp length were simulated for the entire length of each reference sequence in the modified MEGARes v2 database using a sliding window of size 150 bp with a 25bp shift. For the even mutations dataset, each simulated read was artificially mutated at fixed intervals to a random alternative nucleotide (different from the reference sequence), for a number of mutations ranging from 0 to 16 (0% to 10.6% mutation rate). Likewise, for the random mutations dataset, each simulated read was artificially mutated at random locations along the read to a random alternative nucleotide, for a number of mutations ranging from 0 to 16. Each classifier (BWA-MEM, Kraken2, and WarpNL) was then run against

this dataset using the modified MEGARes v2 database as reference. While these datasets were too large to provide as supplementary data, the scripts and files used to generate these datasets are provided in the WarpNL publication code repository (S2 Code).

2.6.6 Methods for Dantas functional metagenomic data analysis

Two functional metagenomic datasets from the Dantas group were used as gold standard data: Soil and Pediatric [67, 68]. In short, functional metagenomics for antimicrobial resistance (AMR) characterization involves cloning fragments of metagenomic DNA into antibiotic susceptible bacterial vectors, then growing these susceptible vectors on antibiotic laden agar. The colonies that grow are hypothesized to have had a gain of AMR function from the cloned metagenomic DNA fragment, which should be highly expressed in these bacteria by plasmid design. A colony from the antibiotic-laden agar is then selected and sequenced, resulting in a sequence dataset that should contain a high percentage of reads with an AMR gene providing resistance against the antibiotic with which the agar was laden. However, some sequences may also belong to the bacterial vector genome. Regardless, since the sequences are from phenotypically resistant bacterial colonies with known antibiotic resistance profiles, these data represent the best gold standard benchmark against which to test AMR sequence classifiers, despite some of the sequences coming from the bacterial vector.

The Soil dataset (NCBI BioProject PRJNA215106) contains 219 samples of Illumina paired-end sequencing data with an average sequence number of 1.98 million per sample. The Soil dataset contains functional metagenomic fragments isolated from soil samples as previously described [68]. The Pediatric dataset (NCBI BioProject PRJNA244044) contains 169 samples of Illumina paired-end sequencing data with an average sequence number of 1.12 million per sample. The Pediatric dataset contains functional metagenomic fragments isolated from pediatric fecal samples as previously described [67]. All classifiers (BWA-MEM, Kraken2, and WarpNL) were run against these datasets, and results were subsequently analyzed using custom R scripts. Each read pair in the Soil and Pediatric datasets was considered to be a single observation (single count) for purposes of

analysis. Observations were considered to be classified "on-target" if their Class-level MEGARes annotation matched the phenotypic AMR label for the Dantas datasets. All scripts, metadata, and NCBI sequence accessions for the Soil and Pediatric datasets are provided in the WarpNL publication code repository (S2 Code).

2.7 Supporting information

2.7.1 Supplementary code and data

- WarpNL source code: https://github.com/lakinsm/nocturnal-llama
- WarpNL publication code: https://github.com/lakinsm/nocturnal-llama-publication
- Dantas Pediatric dataset: BioProject PRJNA244044
- Dantas Soil dataset: BioProject PRJNA215106

2.7.2 Software versions used in this work

- Burrows-Wheeler Aligner v0.7.17 [64]
- Kraken2 v2.0.8-beta [16]
- WarpNL v1.0.0

Chapter 3

GPU-accelerated, distributed computing enables sequence classification using a high-dimensional naive Bayes approach

3.1 Introduction

The naive Bayes algorithm is one of the earlier successful uses of machine learning for the purpose of mapping genetic sequences to taxonomic clades. Its use in the Ribosomal Database Project (RDP) enabled supervised classification of 16S ribosomal gene sequences during microbiome bioinformatics analysis [4]. The problem of 16S gene classification is already constrained by the molecular technique of polymerase chain reaction (PCR), which selectively amplifies a genetic sequence target. This amplified target is then typically sequenced on a high throughput sequencing platform, producing genetic sequences that usually span 150-300 nucleotides of the 16S gene but can span up to the full length of the gene (1,550 nucleotides) [22, 90]. Therefore, a genetic sequence classifier in this context has the advantage of knowing that the genetic sequence belongs to a 16S gene but must then classify the 16S gene sequence into one of many taxonomic clades according to a reference sequence database.

While knowing the target ahead of time poses an easier problem than general sequence classification, the short length of the typical 16S amplicon makes this supervised classification task difficult. Often, a difference of one or two nucleotides can change the membership of the genetic sequence from one clade to a substantially different one, from a phylogenetic perspective [91]. The fact that naive Bayes has been successfully applied in this context suggests that it is capable of discerning small nucleic acid differences for taxonomic profiling using short sequence inputs. Because of this success on a constrained but difficult problem, we hypothesized that naive Bayes

could be extended to accurately perform genetic sequence classification in the general case (not constrained to 16S genes).

We anticipated that classification in the general case would require three critical extensions of an RDP-esque classifier: use of a secondary classifier to perform ingroup versus outgroup classification, expansion of the underlying feature space, and the use of accelerated computing techniques like parallelism and graphics processing units (GPUs) to speed computation. Outgroup classification is a requirement for determining whether a sequence is related enough to the reference sequence database to perform classification to begin with. For 16S classification, the target is known, and outgroup classification is therefore not required; however for the general case, it is necessary to prevent false positive classifications. We solved this problem by combining a *de novo* clustering algorithm using a Gaussian mixture model with a supervised outgroup sequence classifier using a support vector machine. Then, sequences that are classified as ingroup are encoded as described below and classified using the generalized naive Bayes approach.

For our naive Bayes implementation, we utilized an extended feature space: the RDP classifier based their sequence encoding on subsequences of length 8 nucleotides, while ours uses 10 nucleotides. This expands the feature space by 16 times the size of the RDP classifier, allowing for less feature overlap between training classes; this is important since the general classification problem involves many more target classes than just classifying the 16S ribosomal subunit, which is only present in prokaryotic organisms. However, due to the increase in state space, our classifier requires substantially more computational time to perform the naive Bayes calculation, since any increase in the state space is multiplied by an exponential factor of 3 during the matrix multiplication step used to perform the classification itself. Therefore, we additionally leveraged various levels of computational parallelism to increase the naive Bayes algorithm performance, including SIMD directives, multi-threading, work load scheduling and balancing, and hardware acceleration using GPUs and distributed computing. Overall, this allowed for our generalized naive Bayes classifier to perform with comparable resource utilization to state-of-the-art classifiers currently in widespread use. Below, we outline the various critical implementation features of our approach

and additionally make the source code publicly available (https://github.com/lakinsm/nocturnal-llama).

3.2 Feature encoding and optimizations

3.2.1 Input sequence description

The data are assumed to be nucleotide sequences of length L consisting of UTF-8 encoded standard IUPAC bases A, C, G, and T in capital letters. Data used for training WarpNL (the reference database) should be in standard FASTA format, while the data to be classified (query data) should be in standard FASTQ format. Ambiguous nucleotides from the extended IUPAC alphabet are converted at random to one of the standard bases encoded by the ambiguous nucleotide, and the RNA nucleotide uracil is converted into its DNA equivalent thymine.

3.2.2 Multinomial *k*-mer encoding

For the WarpNL multinomial feature encoding, *k*-mers are encoded in the typical way: for each input sequence of length L, contiguous subsequences of length k are considered as *k*-mers, and each *k*-mer window is shifted by one nucleotide during encoding, resulting in a total of $L-k+1$ overlapping *k*-mers. *k*-mers are encoded using the bit shift algorithm described below (Algorithm 1), which leverages the fact that the UTF-8 encoded capital letters A, C, G, and T have two bits in the second and third position that uniquely describe them. The second and third bits are extracted using a bit mask and shifted into an unsigned integer, resulting in the mapping of each *k*-mer into a unique integer in the range of $[0, 4^k)$. The first *k*-mer is encoded in $O(k)$ steps, and each subsequent nucleotide is encoded by shifting the integer bit string over to make space for two new bits, resulting in an algorithm that is both $O(L)$ and that can take advantage of bit-related hardware optimizations that are available to modern processors. Aside from the matrix multiplications encountered during naive Bayes classification, this algorithm is the runtime bottleneck of WarpNL, as determined by function profiling. Therefore, it was deemed important to heavily optimize this routine where possible.

```
// For sequence s and feature vector v
unsigned int kmer_shift_template = 0;
for(int i = 0; i < (k - 1); ++i) {
    kmer_shift_template <<= 2;
    kmer_shift_template |= 3;
}
unsigned int kmer = 0;
for(int i = 0; i < k; ++i) {
    kmer <<= 2;
    kmer |= (s[i] >> 1) & 3;
}
v[kmer] += 1;
for(int i = k; i < s.length(); ++i) {
    kmer &= kmer_shift_template;
    kmer <<= 2;
    kmer |= (s[i] >> 1) & 3;
    v[kmer] += 1;
}
```

Listing 3.1: Algorithm 1. An efficient bit shift algorithm to encode the UTF-8 encoded nucleotides A, C, G, and T into base two-bit *k*-mer representations. This algorithm leverages the fact that these letters have bits that uniquely encode them in their second and third bit positions (A=1000001, C=1000011, G=1000111, T=1010100). By shifting the bitstrings of these letters right by one position, the resulting first two bits uniquely encode them into a two-bit representation (A=00, C=01, G=11, T=10). Using this, the first *k*-mer in sequence s is encoded by the second for loop in Algorithm 1 by shifting the *k*-mer bitstring left by two bits and encoding the two bits from the nucleotide into the newly cleared bit positions at the beginning of the *k*-mer bit string. The feature vector $\boldsymbol{v}$ (of integers) is then incremented by one at the position specified by the *k*-mer's integer representation, in the range of $[0, 4^k)$. For every subsequent *k*-mer after the first one, the *k*-mer bitstring is shifted left by two bits and encoded with the next nucleotide into the newly cleared first and second bit positions, such that work is not repeated for every *k*-mer. The *k*-mer shift template is a bit mask where the first $2(k-1)$ bits are hot, resulting in removal of the last two bits of a *k*-mer bitstring when used with a bitwise and operator. This template is used to clear the last two bits in a *k*-mer bitstring, such that it can be shifted left to make room for a new nucleotide two-bit representation. This process is repeated until the end of the sequence s is reached.

3.2.3 Gallager *l*-mer encoding

The WarpNL Gallager feature encoding, as described in Chapter 2, uses the same bit shift algorithm as outlined in Algorithm 1. However, *l*-mers are additionally extracted from each *k*-mer in an inner loop. This extraction of bits is most efficiently performed using the Parallel Bits Extract (PEXT) instruction available on certain Intel chipsets [92]. Given a bitmask where bits to be extracted are set to one, the PEXT instruction extracts the value of those bits from another bit string and right-aligns them, resulting in a unique integer in the range of $[0, 4^l)$. This setup is analogous to the *k*-mer encoding described above, at the level of *l*-mers instead. We were only able to take advantage of this optimization due to the use of the above algorithm for *k*-mer encoding, further emphasizing its value in this encoding scheme. A complete example is available in the WarpNL source code under the SequenceEncoder class object.

3.2.4 Lidstone smoothing and log-transform

The result of the feature encoding steps are vectors of non-negative integers of length equal to the dimensionality of the features encoded, either 4^k for multinomial encoding or 4^l for Gallager encoding. To transform a count vector into a probability distribution, it must be normalized to sum to 1. For the exponentially high dimensional spaces used by WarpNL, simply dividing by the sum of these vectors can result in floating point underflow. Therefore, a natural log transform is applied to the vectors prior to normalizing them to the unit simplex, which utilizes the fmath.h C++ header routines to accelerate the log transform. Additionally, Lidstone smoothing (adding a constant pseudocount) is applied to each element in the vector during the log transform, such that: all resulting vector elements are defined, all features have non-zero probability, and the rank order of all feature probabilities is preserved. The primary benefit of Lidstone smoothing in a naive Bayes context is that all features have non-zero probability [88]. This allows for features not seen in the training dataset to still be effectively included in the classification decision process; we discuss the importance of this point in the following section dedicated to naive Bayes classification.

3.3 Naive Bayes classification

Naive Bayes has been successfully used for genetic sequence classification in the past, particularly for more constrained problems like 16S ribosomal gene classification in the context of microbiome analysis [4]. However, these previous publications do not delve into details regarding the nuances of implementing a naive Bayes classifier for genetic sequences. Here, we discuss the components of performing naive Bayes classification in this context and cover a few behaviors of this approach that are likely known to machine learning practitioners but which are rarely mentioned in the literature. We then cover how we implemented naive Bayes classification in WarpNL in a way that was appropriate for high performance computing.

One such nuance is that naive Bayes classifiers ultimately rely on small differences between classes to perform classification. This is a result of its use of the maximum *a posteriori* estimate, which performs well in determining the most likely class of assignment but does not perform well when comparing relative *a posteriori* estimates between classes. Said another way, the naive Bayes MAP estimate is often the "correct" class of assignment, however the actual value of the MAP estimate tells us little about how confident that assignment is relative to alternative class options. This is particularly important when performing multi-way classification with many class options available to the classifier. The more options there are, the less likely the classifier is to be correct by chance, making multi-way classification much more difficult than a task like binary classification.

In practice, the difference is often small between the assigned class (MAP estimate) and the next highest *a posteriori* estimate. In some cases where classes have substantial feature overlap, these small differences may result in the incorrect class having the MAP estimate, while the correct class has the second highest estimate. Yet, because the MAP estimate is not a posterior probability, it cannot be used directly to produce a confidence metric for when such a misclassification might arise. Therefore, we present below a derivation of an information criterion that can be used to produce a confidence metric for this scenario, which is implemented in WarpNL.

This issue of the assigned class having the MAP estimate by only a small margin is particularly pronounced in high dimensional, sparse feature spaces. In Chapter 2, we showed that feature encoding strategies for genetic sequence classification produce high dimensional feature spaces that are sparse in practice. Lidstone smoothing combined with the presence of low integer values in a sparse, high-dimensional feature space means that features actually observed in the training data often have only one additional count than "pseudo-features" that are only "observed" as a result of the smoothing process. Therefore, the training distribution is very flat in practice, which leads to these small differences between *a posteriori* estimates. However, if one attempts to use naive Bayes in high dimensional feature spaces without additive smoothing, the resulting classifier performs so poorly that it isn't even worth discussing here. It is therefore clear that naive Bayes performs well in genetic sequence classification by using the small differences in relatively flat probability distributions to break ties between classes.

This behavior is somewhat surprising but interesting in that it has implications for how various smoothing techniques are expected to perform in a genetic sequence classification problem. Additive smoothing typically performs well for high dimensional, sparse distributions with small feature counts, because the probability distribution is so flat that corrections need not be performed for document frequency across the class corpus. For example, TF-IDF (term frequency inverse document frequency) smoothing corrects feature counts according to how many documents (in this case genomes) are observed for a given class, since the more documents that are observed, the more confident we are that features are true for this particular class [93, 94] Likewise, classes with few documents (observed genomes) should be considered as lower confidence choices during classification, because we have not sampled the feature space as well for that class. However, if one does not correct across the corpus, normalization to the unit simplex results in the low confidence class having the same weight as the high confidence class. Techniques like TF-IDF smoothing can be used to discount or weight the classes according to how many observations were used to produce their probability distributions, usually resulting in better naive Bayes classifier performance. But for genetic sequence classification, the distributions are so flat due to their high dimensionality

that no such corpus-level corrections are typically needed. Therefore additive smoothing, which does not account for corpus differences, works fine in practice. Term frequency corrections can, however, still be useful.

Below, we provide derivations for naive Bayes classification using Lidstone smoothing and the previously discussed information criterion for comparison of MAP estimates resulting from naive Bayes classification. Both of these strategies are implemented in WarpNL and can be found in the relevant class objects as listed in each section.

3.3.1 Derivations

Naive Bayes classification using the inner product

Let $\boldsymbol{t_r} = \{t_{1r}, t_{2r}, \ldots, t_{Dr}\}$ be a vector of non-negative integers associated with a query sequence read or read-pair r that we wish to classify using naive Bayes, such that $N = \sum_{d=1}^{D} t_{dr}$. t_{dr} represents the count of k-mer d within sequence read r. Accordingly, we can construct a query matrix of counts for each feature (k-mer), $\underset{R\times D}{\boldsymbol{T}}$, where R is the number of query sequence reads, and D is the dimensionality of the feature space. Moreover, and utilizing a training sequence data set or database, we can construct a matrix, $\underset{D\times C}{\boldsymbol{X}}$, with columns on the unit simplex ($\boldsymbol{x_c} = \{x_{1c}, x_{2c}, \ldots, x_{Dc}\}$ and $\sum_{d=1}^{D} x_{dc} = 1$), where C is the set of classes to which a query sequence read can be classified. Assuming that each $\boldsymbol{t_r} \in R$ is independent, we consider a single query sequence read at a time for classification. Our goal is to identify the most likely class $C = c$ where $\boldsymbol{t_r}$ belongs. Applying Bayes theorem,

$$P(C = c|\boldsymbol{t_r}, \boldsymbol{X}) = \frac{P(\boldsymbol{t_r}|\boldsymbol{X}, C = c)P(C = c|\boldsymbol{X})}{P(\boldsymbol{t_r}|\boldsymbol{X})} \tag{3.1}$$

The values of $\boldsymbol{t_r}$ are fixed, and therefore the denominator is constant across classes. In addition, assuming that the classes are also independent, the problem reduces to comparing likelihood of $\boldsymbol{t_r}$ between the different classes $c \in C$,

$$P(C = c|\boldsymbol{t_r}, \boldsymbol{x_c}) \propto P(\boldsymbol{t_r}|\boldsymbol{x_c}, C = c)P(C = c|\boldsymbol{x_c}) \tag{3.2}$$

Where $P(\boldsymbol{t_r}|\boldsymbol{x_c}, C = c) \equiv P(\boldsymbol{t_r}|\boldsymbol{x_c})$ is the likelihood of observing $\boldsymbol{t_c}$ given the class $C = c$ that is characterized by the vector of proportions $\boldsymbol{x_c}$ associated with that class, and $P(C = c|\boldsymbol{x_c}$ is the prior probability of belonging to class c. The likelihood $P(\boldsymbol{t_r}|\boldsymbol{x_c})$ can be modeled using a variety of probability distributions, with the most common being the multinomial, which we use here, or Dirichlet multinomial [95]. A prior over the classes can be used, however here we assume that all classes have equal prior probability. Accordingly, utilizing the assumption that the counts in $\boldsymbol{t_r}$ are multinomially distributed and that $P(C = c|\boldsymbol{x_c})$ is equal for each class c, the above reduces to,

$$P(C = c|\boldsymbol{t_r}, \boldsymbol{x_c}) \propto \prod_{d=1}^{D} P(t_{dr}|\boldsymbol{x_c}, C = c) \tag{3.3}$$

Our goal then reduces to finding the class c with maximum *a posteriori* (MAP) estimate (gives the maximum likelihood of the query data), also called the naive Bayes (NB) estimate, across the classes $c \in C$ given by:

$$c_{\text{NB}} \propto \underset{c \in C}{\text{argmax}} \prod_{d=1}^{D} P(t_{dr}|\boldsymbol{x_c}) = \underset{c \in C}{\text{argmax}} \prod_{d=1}^{D} x_{dc}^{t_{dr}} \tag{3.4}$$

Taking the natural log of both sides after adding a Lidstone pseudocount α [88] results in,

$$c_{\text{NB}} \propto \underset{c \in C}{\text{argmax}} \sum_{d=1}^{D} t_{dr} \ln(\alpha + x_{dc}) \equiv \underset{c \in C}{\text{argmax}}\ \boldsymbol{t_r'} \cdot \ln(\boldsymbol{x_c} + \boldsymbol{\alpha}) \tag{3.5}$$

where $\boldsymbol{t_r'}$ is the transpose of $\boldsymbol{t_r}$. Which we can apply to classify all observed reads $r \in R$ in the query matrix $\boldsymbol{T}$, this is the definition of the matrix inner product,

$$c_{\text{NB}} \propto \underset{c \in C, r \in R}{\text{argmax}}\ \boldsymbol{T} \cdot \ln(\boldsymbol{X}) \tag{3.6}$$

Contained in the above derivation are several important points: 1) since the c_{NB} estimate is relative due to dropping the $P(\boldsymbol{t_r}|\boldsymbol{X})$ term from the denominator of Bayes theorem, these *a posteriori* estimates are not easy to compare in order to assess how much better the class producing

the NB estimate is than the class producing the second or third highest *a posteriori* estimates; 2) naive Bayes uses many assumptions that may not be true in practice, including independence of the features (*k*-mers), independence of the data draws ($\boldsymbol{t}$ counts), and equal probability of observing each class; and 3) the general structure of the above derivation reduces to a simple matrix multiplication, which can be leveraged for efficient computation. Below, we expand on several of these points to show how naive Bayes can be adapted into an efficient, modern framework.

Information criterion for comparison of *a posteriori* estimates

As shown in the derivation of the naive Bayes MAP estimate, the *a posteriori* naive Bayes estimates cannot be directly compared as posterior probabilities. Below, we provide a criterion for comparison of *a posteriori* naive Bayes estimates that is analogous to the log odds ratio and can be interpreted in the same way [96]. This criterion produces roughly normal distributions that have clear discriminatory power between true positive and misclassifications up to around 10% mutation rates on simulated short read data (Figure 3.1). Negative values of the WarpNL information criterion (WarpNL IC) are also clearly indicative of a misclassification and suggest that the next highest *a posteriori* estimate is likely the more accurate choice. These data were produced using the even and random mutation datasets that were introduced in Chapter 2.

Following the above notation, let c_{NB1} and c_{NB2} be the highest and second highest *a posteriori* naive Bayes estimates obtained by classification of $\boldsymbol{t}$. Let $\boldsymbol{x_1} = \{x_{11}, x_{21}, \ldots, x_{D1}\}$ and $\boldsymbol{x_2} = \{x_{12}, x_{22}, \ldots, x_{D2}\}$ be the training data probability vectors on the unit simplex that correspond to c_{NB1} and c_{NB2}. By the above proof of naive Bayes, $c_{\text{NB1}} = \sum_{d=1}^{D} t_d \ln(x_{d1})$, and $c_{\text{NB2}} = \sum_{d=1}^{D} t_d \ln(x_{d2})$. A test vector $\boldsymbol{t}$ with data that perfectly align to training data probability vectors $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ would have the maximum expected values of *a posteriori* estimates $c_{\text{NB1}_{\text{max}}} = N \sum_{d=1}^{D} x_{d1} \ln(x_{d1})$ and $c_{\text{NB2}_{\text{max}}} = N \sum_{d=1}^{D} x_{d2} \ln(x_{d2})$. The WarpNL log odds ratio test (information criterion) would then be,

$$W_{\text{lod}} = (c_{\text{NB1}} - c_{\text{NB1}_{\text{max}}}) - (c_{\text{NB2}} - c_{\text{NB2}_{\text{max}}}) \tag{3.7}$$

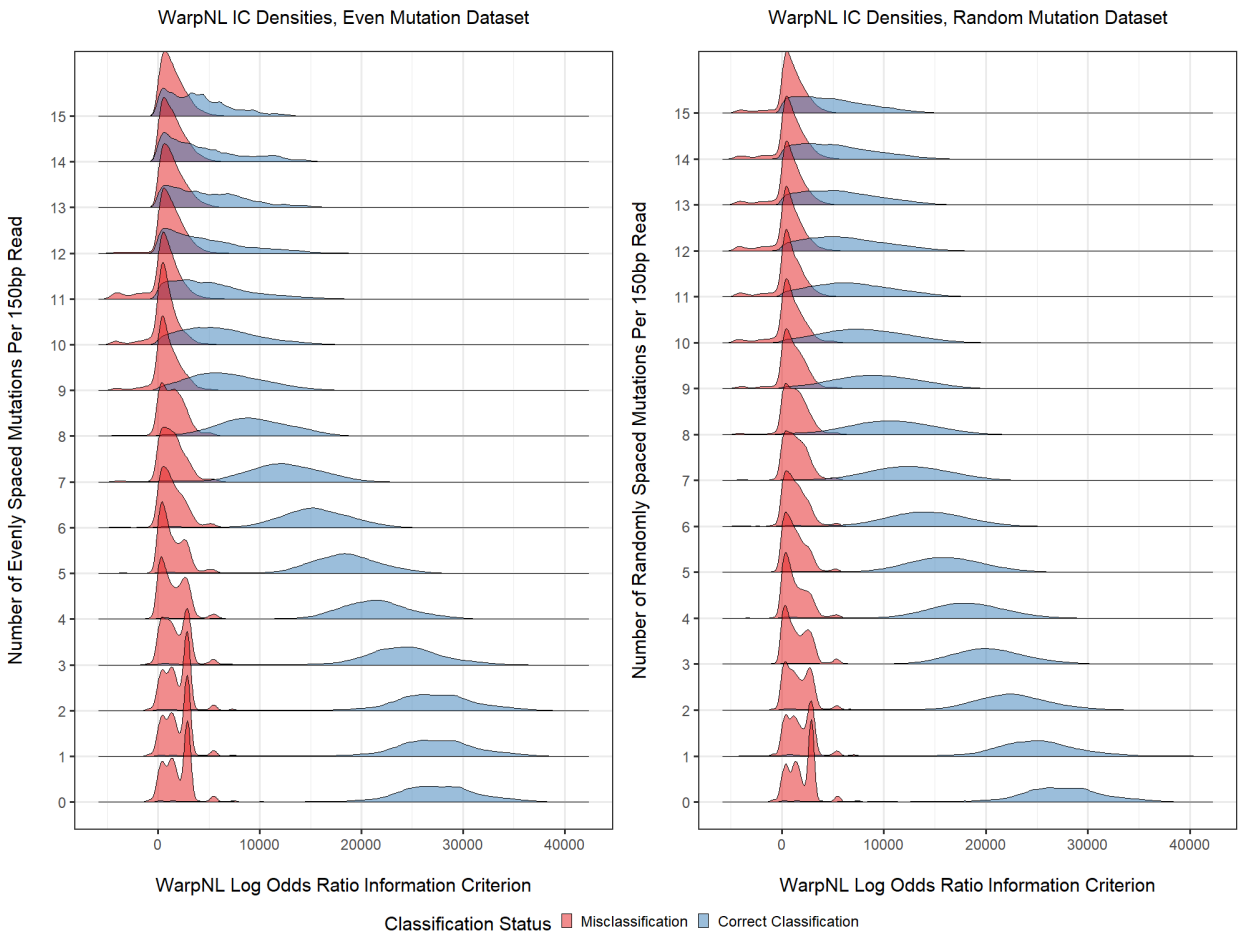

Figure 3.1: The WarpNL log odds ratio information criterion produces roughly normal densities that clearly discriminate between misclassifications and correctly-classified sequences. This criterion can be used as a measure of confidence regarding the accuracy of the MAP estimate relative to its next highest *a posteriori* estimate. Misclassified reads nearly always have a low value of the WarpNL IC (less than 8000). Correctly classified reads that are a small number of mutations away from the database have much higher values of the WarpNL IC, but the criterion decreases as the mutation rate increases. Above around 10% mutation rate, correctly classified reads still have higher values than misclassified reads, but the difference isn't as distinguishable. Negative values of the WarpNL IC nearly always indicate a misclassification, where the second highest *a posteriori* class is more likely to be the correct classification. These data were produced using the even and random simulated mutation datasets introduced in Chapter 2.

3.3.2 Data structures and matrix product

The implementation of naive Bayes in WarpNL consists of three matrix data structures and the GPU-accelerated matrix multiplication. These data structures were implemented as C++ classes in an object oriented programing framework. The three matrices are: 1) the left-hand matrix (class EmpiricalMatrixNL) that holds the empirical, non-negative integer count data to be classified (analogous to $\boldsymbol{T}$ in the derivation), 2) the right-hand matrix (class ProbabilityMatrixNL) that holds the training data probability vectors on the unit simplex (analogous to $\boldsymbol{X}$ in the derivation), and 3) the result matrix (class ResultMatrixNL) that stores the *a posteriori* estimates. The empirical and probability matrices are encoded in CUDA half precision (16-bit floating point) to reduce the memory footprint and allow for CUDA optimization of the matrix multiplication routine. The result matrix is encoded in full precision (32-bit floating point) to prevent numerical underflow from occurring during the matrix multiplication routine. The multiplication itself is the GemmEx cuBLAS library routine for mixed-precision matrix multiplication, which can take half precision matrices as input and produce a full precision matrix as a result.

Use of the half precision data type reduces the memory footprint of the empirical and probability matrices by half. This is substantial, since these matrices have one dimension of size 4^k, making them the space complexity bottleneck for WarpNL. Additionally, Pascal and later NVIDIA GPU architectures have hardware capabilities that accelerate the matrix multiplication when performed with mixed or half precision, resulting in a further decrease in runtime. Since WarpNL is constrained in both space and time complexity, it was deemed important to pursue such optimizations. In fact, without these optimizations, WarpNL would not be efficient enough to compete with other modern classifiers, which is historically why naive Bayes approaches fell out of favor in bioinformatics over the last decade. However, with the optimizations present in WarpNL, it is only 100-fold slower than very efficient classifiers like Kraken2, when used with sufficiently advanced NVIDIA GPU architectures and Intel CPU architectures. This level of efficiency makes it suitable even for large datasets like the Soil functional metagenomic dataset introduced in Chapter 2, which took only 48 hours for WarpNL to classify.

3.4 Accelerated and distributed architectures

WarpNL achieves its runtime performance through the use of GPU-acceleration, efficient CPU data structures and algorithms, and the ability to dynamically allocate resources using active workload monitoring. Having a dynamic load-balancer integrated into the software is critical to its performance, since WarpNL performs a variety of tasks with varying input size and time complexities. For example, the routine used to train the classifier performs K-fold statistical cross-validation, which requires re-allocation and computation of the large probability matrix at every iteration. However, the individual folds of the cross-validation process are independent and therefore able to be parallelized across CPU cores. Since one of the parameters optimized by the cross-validation is the choice of k, the feature size, these probability matrices vary from small to extremely large at the rate of 4^k, meaning WarpNL must determine dynamically how much memory the system is capable of using at any given time. Active monitoring of the memory load must be performed for both the system random access memory (RAM) and the GPU RAM for the naive Bayes cross-validation process.

However, this naive Bayes cross-validation procedure is only one of the multiple tasks that WarpNL must load-balance, depending on the pipeline being used. The full list of considerations that WarpNL must account for are as follows. For the training pipeline: 1) grid cross-validation of the naive Bayes feature encoding type (multinomial or Gallager), the feature size (*k*-mer and/or *l*-mer size), the Gallager row weight, and the smoothing constant α; 2) optimization of the bivariate Gaussian mixture model for *de novo* clustering of outgroup sequences during the naive Bayes cross-validation pipeline; and 3) optimization of the decision boundary and grid cross-validation of the bias λ hyperparameter for the outgroup support vector classifier. For the classification (test) pipeline: 1) distributing input sequences to the the CPU cores for outgroup classification and to the GPU devices for naive Bayes classification; and 2) aggregation of the final result files to complete the classification process.

The tasks for the training pipeline involve work of varying size, while the tasks for the test pipeline are relatively constant in size. This means that WarpNL must have strategies in place

to load-balance for a variety of input tasks, which is most easily done via active monitoring of the system resources. Below, we outline the structure of WarpNL's distributed and accelerated optimizations and describe a selection of the components that drive WarpNL's active monitoring and load balancing system.

The schematic for WarpNL's load-balancer is outlined in Figure 3.2. This diagram includes the WarpNL load balancing components, the input data structures, and the computational resources for both workstation (single node) and distributed computing (multi-node). Since the limiting resource is GPU memory, the GPU resources are the focus of this schematic, however CPU resources are also utilized and monitored by WarpNL. In Figure 3.2, the process is launched on the the workstation or head node of a distributed cluster. This process simply starts the WarpNL pipeline, parses the command line arguments and settings, and actively waits for the computation to complete. For distributed systems, this process would be the master communication process and would communicate via message passing interface (MPI) with the distributed compute nodes. When run locally on a single node or workstation, all parent and children processes run concurrently on the same resource.

The master instance controls launching of node instances and job producer instances. These processes are efficiently managed using a dispatch queue where process threads are spawned but only utilized when the job queue has waiting work to be completed; otherwise the process worker threads hibernate. This dispatch queue system is very efficient and is commonly used for battery-powered devices like phones, where active threads may consume valuable resources, which is undesirable behavior. The first action taken by the master instance is to query the distributed compute nodes or local system to gather information about resource availability and current utilization. Then, the master instance launches job producers. The job producers parse the input data into working chunks according to the resources available on the compute nodes or processes and enqueue these chunks into the master work queue. The master work queue then greedily distributes them to nodes using a max priority queue, locally decrementing the node resources until they are

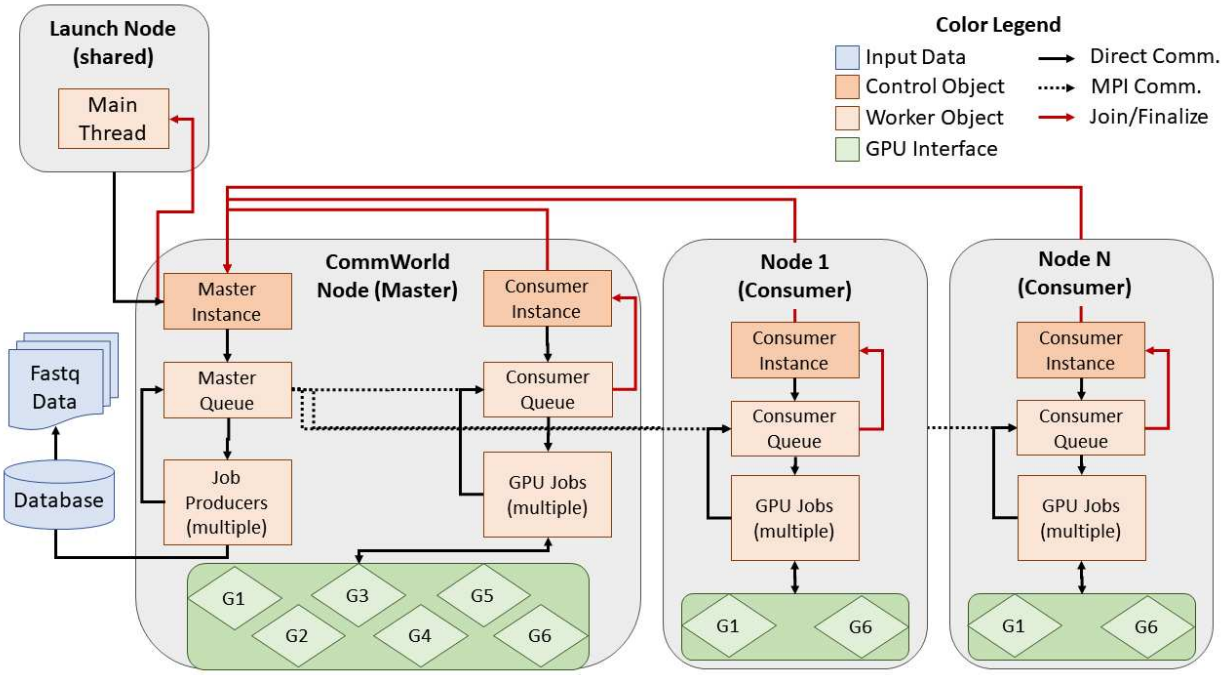

Figure 3.2: The WarpNL distributed load-balancer uses an object oriented design, where each class object manages a worker thread that performs a discrete task. The main thread launches the master instance, which manages job creation and system-wide resource monitoring/scheduling. The master instance in turn launches consumer instances that manage intra-node resources and scheduling, particularly with regard to the GPUs. GPU jobs execute the assigned task and return to the consumer instance for additional work or to finalize. Likewise, the consumer instances finalize back to the master instance when all work assigned by the master instance has been completed. All queue structures are managed using dispatch queues, which cause threads to sleep instead of spin while idle, preventing unwanted consumption of resources. In this way, WarpNL splits large datasets into highly parallel tasks, which results in efficient computation.

consumed. The master queue then waits via MPI or mutex lock for the nodes to indicate that they have completed computation and have free resources.

The node instances receive the jobs from the master instance and place them into another priority queue for distribution to the GPU devices. Each node instance is only aware of its own local resources and is in charge of actively managing jobs on those resources. A second dispatch queue is used by the node instance to dispatch one worker thread per job; each job is assigned a specific set of resources that it can utilize, and the job worker thread manages computation then returns to the node instance dispatch queue when complete. The job worker threads fill the data structures outlined above for the various pipelines and submit their jobs to the resources. The NVIDIA GPU devices have their own onboard scheduler and load-balancer, which handles allocation of the job arrays onto the device itself. WarpNL is capable of taking advantage of streaming computation on the more modern NVIDIA architectures, so multiple jobs can be submitted to the same device. This secondary aspect of parallelization allows data transfers across the PCIe bus to be completed while compute is happening on the device, effectively overlapping data transfer and compute to speed up computation on serially submitted jobs. Since WarpNL is largely memory constrained, overlapping of data transfers is an important part of efficiency, since a large amount of data must be transferred to the device for each job.

When jobs are completed, they report back to the node instance dispatch queue, which updates the node watcher's available resource count. This node resource data structure is an important component that is checked periodically by the master instance watcher, so that the master instance can distribute additional jobs if needed. During computation, the node instance does not initiate communication with the master instance but does respond to requests to be updated by the master instance. This allows resources on the node instance to be focused on managing local compute, while the master instance is focused on distributing jobs to the nodes. Therefore, there is no local wait time on the node instances other than the time it takes for the master instance to push new jobs to the nodes.

When all computation is complete, the node instances notify the master instance and shut down their dispatch queues and watcher instances. In turn, when all node computation is complete, the master instance shuts down its resource watcher and proceeds into aggregation of the distributed result files. Serial CPU-based tasks are then completed on the master instance. This includes optimization of the support vector machine classifier for the training pipeline and result file aggregation and formatting for the test pipeline. Finally, results are output to the user.

In theory, this setup is capable of distributed computation on any number of nodes. The inter-node (local) data structures were all successfully implemented into WarpNL and include logic for splitting large data matrices across GPU resources. This splitting of large resources could be extended to MPI-based multi-GPU compute if desired. While this setup includes schema for MPI implementation, we did not make use of MPI in WarpNL, since we found that most computation can be completed using single-node or workstation resources and parallelized using standard schedulers on distributed clusters. However, if one had a dataset that demanded MPI distributed computing, WarpNL's source code could be a good starting point to implement such a program. Additionally, WarpNL is staged such that it could take advantage of further data transfer and GPU RAM optimizations like NVLink compute, however these optimizations were determined to have minor impacts on efficiency and therefore were not implemented.

3.5 Statistical cross-validation

Cross-validation is the iterative procedure of excluding a portion of the training data from the training set to use as a test or validation set for parameter optimization during the machine learning process. Typically, the portion of the training set excluded is non-overlapping and is between 1% and 25% of the data at a time. Each iteration is commonly called a "fold," and folds are typically chosen to be equal in size, resulting in a procedure called "K-fold cross-validation" [89]. Inherent in the choice of fold size is a bias-variance tradeoff: the smaller the fold size (fewer observations per fold), the more biased the parameter estimates produced, and the larger the fold size (more observations per fold), the more variance the parameter estimates will have. In typical machine

learning tasks, for example in the domain of natural language processing, it is often realistic to choose folds of equal size, which helps to ensure that the parameter estimates are within a certain degree of the bias-variance tradeoff as determined by the analyst. If, however, the training set has highly unequal numbers of observations per class (class imbalanace), then either a small fold size must be used across all classes to maintain and equal fold size, or variable-sized folds must be used.

In the case of genetic sequence classification, it is common for training datasets to have large class imbalance, with many classes having few observations and few classes having many observations. This is true of the database used in Chapter 2: the MEGARes v2 antimicrobial resistance database. For WarpNL, the time it takes to build the training matrix and compute on it for values of $k > 9$ is often more than several minutes, resulting in potential days of computing if the analyst chooses the fold size to be too small with many parameters combinations that need to be explored. This is true even with the high degree of parallelization that WarpNL allows for the cross-validation process. Therefore, it is desirable for WarpNL to use as large a fold size as possible while still obtaining viable parameter estimates. However, the MEGARes genetic sequence database contains many classes with only one or two observations and a good number of classes with very few observations. Therefore, for the sake of efficiency and obtaining accurate parameter estimates, we chose to use variable fold sizes, breaking from the traditional approach of K-fold cross-validation.

WarpNL's strategy for implementing cross-validation is to attempt to split class observations into K folds, as specified by the user. However, if the number of observations in a given class is fewer than K, WarpNL tries to perform leave-one-out cross-validation for that class (leaving a single observation out per fold). If the class node has only one observation, then the class is excluded from the cross-validation procedure, since it is effectively useless for determining optimal model parameter values. WarpNL therefore performs variable K-fold cross-validation to determine optimal naive Bayes parameters and to produce an outgroup test set for later training the

outgroup support vector machine classifier. This latter process, for training the outgroup classifier, is described in detail below.

3.6 Outgroup classification

Determining whether a sequence is "close enough" to the training dataset to classify it using only information contained the sequence itself is perhaps one of the most challenging problems in genetic sequence classification. We argue that this process is so complex and drives at such deep problems in bioinformatics that producing a viable solution to this problem that isn't a just an arbitrary heuristic threshold should be considered a success, particularly in the context of machine learning. As described in Chapter 2, WarpNL makes use of a Gaussian mixture model (GMM) *de novo* clustering algorithm to re-label sequences as outgroup sequences if they are too distant from the ingroup sequences after being left out during K-fold cross-validation. The data with *de novo* labels after GMM optimization are then used to train a decision boundary for a support vector machine classifier. Both the GMM and SVC optimization routines are described below as they are implemented in WarpNL.

3.6.1 Feature encoding for outgroup classification

For both the multinomial and Gallager encoding methods, a separate value of k is chosen for outgroup classification that is independent of the encoding strategy used for naive Bayes classification. For the work in Chapter 2, we used $k = 12$ for outgroup classification, which would have been prohibitively large for construction of the naive Bayes matrices. However, since we aren't encoding all possible *k*-mers for outgroup classification, using a larger value of k is possible. During construction of the probability matrix using the training data, all observed *k*-mers in the training data are hashed into a set for both the forward and reverse sequences. Then, for all test or left-out sequences, the percentage of *k*-mers that are not present in the training *k*-mer set is recorded for the forward read and reverse read if present. If the reverse read is not present (as in single-end sequencing data), then the forward read percent *k*-mer mismatch is used for both the x_1 and x_2

dimensions as input for the GMM and SVC optimization routines. The result of this process is a two-dimensional dataset of non-negative real numbers in the range of $[0, 100]$, representing the forward and reverse mismatch percentages, respectively.

3.6.2 Gaussian mixture model implementation

A bivariate Gaussian mixture model *de novo* clustering algorithm was chosen because the percent mismatch data described above appeared to be normal in two dimensions. Prior to GMM optimization, data $\boldsymbol{x}_i \in \mathbb{R}^2, i \in 1, 2, \ldots, N$ were labelled as ingroup if they were both a sequence left out as a part of K-fold cross-validation and if $x_1 < 80$ and $x_2 < 80$. Data were labelled as outgroup if they were left out of K-fold cross-validation but either $x_1 \geq 80$ or $x_2 \geq 80$, and all user-provided outgroup sequences were by default considered to be outgroup sequences prior to GMM optimization. The sequences as labelled above served as an initial state for the GMM optimization routine. The initial values for the ingroup mean were set to $\boldsymbol{\mu}_1 = \langle 20, 20 \rangle$ and outgroup mean $\boldsymbol{\mu}_2 = \langle 80, 80 \rangle$. Initial values for both the ingroup ($\boldsymbol{\sigma}_1$) and outgroup ($\boldsymbol{\sigma}_2$) variance (diagonal) were set to $\boldsymbol{\sigma} = \langle 30, 30 \rangle$. The covariance parameters on the off-diagonal of the variance-covariance matrix were fixed at 0 for the entire optimization procedure. This decision was made because if the covariance was allowed to be optimized, the resulting outgroup bivariate Gaussian distribution fit tightly along the $x = y$ line, which produced nonsensical *de novo* labels. Optimization of the GMM class distribution parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ was performed via expectation maximization (EM) of the bivariate Gaussian distributions. The following outlines the basic principles of EM optimization of two-distributional mixture of bivariate Gaussians [97].

In the E-step, the current values of $\boldsymbol{\mu}_1, \boldsymbol{\sigma}_1$ and $\boldsymbol{\mu}_2, \boldsymbol{\sigma}_2$ are used to update the weights $\hat{w}_{ki}, i \in 1, 2, \ldots, N, k \in 1, 2$ for each observed data point. These weights describe the proportion that a given data point "belongs" to each of the two Gaussian distributions and therefore determine how heavily a data point is weighted during parameter updates to each distribution. So as is standard for the EM algorithm, the distributional parameters are fixed in the E-step and used to update the data point weights. The parameters ϕ_1 and ϕ_2 describe the weight of each distribution (average

across the data point weights) are also updated in this step. The data point weights are updated as follows,

$$w_{ki} = \frac{\phi_k}{2\pi|\boldsymbol{\Sigma}_k|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu}_k)\right) \tag{3.8}$$

The new weights are then normalized by their sum across the distributions, such that they sum to 1 for each data point,

$$\hat{w}_{ki} = \frac{w_{ki}}{\sum^k w_{ki}} \tag{3.9}$$

Membership of each data point is then determined to be the larger of the two weights, so if weight $\hat{w}_{1i} > \hat{w}_{2i}$, then data point $\boldsymbol{x}_i$ would be labelled as belonging to distribution 1, and vice versa. In the M-step, the weights are fixed and used to perform an update of the parameters $\phi, \boldsymbol{\mu}, \boldsymbol{\sigma}$ for both distributions.

$$\phi_k^{new} = \frac{\sum_{i=1}^N \hat{w}_{ki}}{N} \tag{3.10}$$

$$\boldsymbol{\mu}_k^{new} = \frac{\sum_{i=1}^N \hat{w}_{ki} \cdot \boldsymbol{x}_i}{\sum_{i=1}^N \hat{w}_{ki}} \tag{3.11}$$

Using the new means as calculated above,

$$\boldsymbol{\Sigma}_k^{new} = \frac{\sum_{i=1}^N \hat{w}_{ki} \cdot (\boldsymbol{x}_i - \boldsymbol{\mu}_k^{new})(\boldsymbol{x}_i - \boldsymbol{\mu}_k^{new})^T}{\sum_{i=1}^N \hat{w}_{ki}} \tag{3.12}$$

However, as described above, the covariance terms in the variance-covariance matrix were fixed at 0 during our implementation of the above EM routine. Loss was calculated using the evidence lower bound of the bivariate Gaussian mixture to guarantee convergence from Bishop 2006 equation 9.74 [97]. At the end of the EM routine, the labels for each data point correspond to which of the two distributions has the higher weight (is more likely to belong to that distribution).

This produces *de novo* labels that are then used as input into the support vector machine described below.

3.6.3 Support vector machine classifier implementation

A support vector machine classifier (SVC) was implemented using the same two-dimensional input feature space described above in this section. A linear decision boundary was used for both efficiency of optimization and efficiency of assigning labels to new data: the linear SVC label assignment can be directly calculated from the equation for the linear decision boundary. For non-linear SVC boundaries that make use of a kernel space, labels must be calculated by comparing the new data point against all of the support vectors, which takes time when the number of support vectors and the number of data points to be classified is large [83]. Since WarpNL aims to classify billions of sequences as efficiently as possible, the SVC classification step must be efficient enough to not slow down the encoding of sequences into the naive Bayes classification process. This ultimately required the use of a linear SVC. The linear SVC did not result in a degradation of classification accuracy, since the boundary between the two classes produced by the bivariate GMM appeared to be roughly linear, making it an appropriate choice for the problem.

The standard derivation for the SVC optimization problem is as follows. Given training data $\{(\boldsymbol{x}_i, y_i)\}_{1 \leq i \leq n}, \boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \{+1, -1\}$, the primal optimization problem is [83],

$$\min_{\boldsymbol{w},b} \|\boldsymbol{w}\|^2 + \lambda \sum_{i=1}^{n} \xi_i^p \quad \text{s.t. } y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, \xi_i \geq 0 \tag{3.13}$$

Where $\boldsymbol{w}$ are the support vector weights, $\boldsymbol{\xi}$ are the slack variables, b is the intercept, and λ (commonly notated as C, but we use λ to avoid confusion with previous derivations) is the bias hyperparameter. Alternatively, the dual formulation arises from applying Lagrangian multipliers to the primal,

$$\mathscr{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 \right] \quad \text{s.t. } \alpha_i > 0 \quad \forall i \in 1, 2, \ldots, n \tag{3.14}$$

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i^T \boldsymbol{x}_j) \quad \text{s.t. } \alpha_i \geq 0 \quad \forall i \in 1, 2, \ldots, n \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0 \quad (3.15)$$

Most SVC implementations optimize the Lagrangian dual formulation using hinge loss, since most optimizations for modern datasets using SVMs involve data of high dimensionality. Therefore, it is often more efficient to find sparse solutions to the Lagrangian dual formulation than it is to optimize the primal. However, we chose to optimize the less commonly used primal formulation using squared loss due to its numerical stability, which produced better decision boundaries for this two-dimensional problem. Since our problem has low dimensionality, this approach did not suffer from the usual efficiency issues involved with optimizing the primal using data in high dimensional space.

Optimization of the primal formulation using squared loss was performed via Newton-Raphson gradient descent with line search [98]. All code for this process was written specifically for WarpNL to ensure maximum efficiency in performing small matrix-matrix products, matrix inversions, and parameter updates during optimization. The working example of the above process is contained in the SupportVectorClassifier class object in the WarpNL source code.

Chapter 4

Fast maximum likelihood estimation and supervised classification for the beta-Liouville multinomial

4.1 Summary

The multinomial and related distributions have long been used to model categorical, count-based data in fields ranging from bioinformatics to natural language processing. Commonly utilized variants include the standard multinomial and the Dirichlet multinomial distributions due to their computational efficiency and straightforward parameter estimation process. However, these distributions make strict assumptions about the mean, variance, and covariance between the categorical features being modeled. If these assumptions are not met by the data, it may result in poor parameter estimates and loss in accuracy for downstream applications like classification. Here, we explore efficient parameter estimation and supervised classification methods using an alternative distribution, called the Beta-Liouville multinomial, which relaxes some of the multinomial assumptions. We show that the Beta-Liouville multinomial is comparable in efficiency to the Dirichlet multinomial for Newton-Raphson maximum likelihood estimation, and that its performance on simulated data matches or exceeds that of the multinomial and Dirichlet multinomial distributions. Finally, we demonstrate that the Beta-Liouville multinomial outperforms the multinomial and Dirichlet multinomial on two out of four gold standard datasets, supporting its use in modeling data with low to medium class overlap in a supervised classification context.

4.2 Introduction

Count data arise in many contexts, including in natural language processing, ecology, and bioinformatics [99–101]. For example, data such as the words in an email could be features that are counted and used to classify each email into a category (e.g. spam or ham). While many prob-

ability distributions can be used to model categorical count data, the multinomial and Dirichlet multinomial (DM) distributions have been shown to perform well and are widely used for this task [102]. The simplicity of these distributions enables their integration with computationally efficient classifiers like naive Bayes, making them popular for large-scale classification tasks. For example, one can obtain estimates of multinomial parameters over any number of features by simple division, circumventing the need for more complicated parameter estimation techniques. Additionally, the Dirichlet distribution is a convenient prior distribution for the multinomial, resulting in "smoother" multinomial estimates that help to prevent overfitting of the training data [103].

However, the multinomial and DM distributions assume strict negative covariance between features, making them ill-suited for modeling data that violate this assumption. Since features often co-occur in practice, these distributions may not be optimal for capturing nuanced feature relationships in categorical datasets, such as semantics in text processing [104]. These distributions also have means and variances for each feature that are linked due to sharing the same parameters, much like the Poisson distribution. As a result, features with truly different variances but the same mean will be modeled identically, which can further confuse classifiers. To address these issues with the multinomial and DM distributions without sacrificing computational efficiency, the Beta-Liouville distribution can be used as a conjugate prior to the multinomial distribution. Utilizing a Beta-Liouville prior gives rise to the Beta-Liouville multinomial (BLM) distribution [105]. The BLM relaxes the previously mentioned assumptions by introducing an additional parameter over the DM distribution, which may improve modeling results.

Recent applications of the BLM distribution have been explored by Bouguila and Fan [102, 106, 107], particularly in computer vision and natural language processing. Bouguila's work has focused on unsupervised clustering and semi-supervised classification using methods such as expectation maximization, variational learning with finite mixture models, and online learning. Bouguila's group has shown that the BLM distribution improves upon the DM distribution for certain datasets where class labels are determined by the learning process, such as in clustering via expectation maximization [106]. However, their work has not applied the BLM distribution

to strictly supervised problems, in which the true class labels are fixed and not determined by the learning process. Additionally, their work does not thoroughly explore the computational efficiency and stability of learning BLM-based models from observed data. In this manuscript, we extend Bouguila's work by applying the BLM distribution to supervised classification, focusing on efficient computation, stability of parameter optimization, and classification power.

Previous work by Sklar [108] provided an efficient implementation of Newton-Raphson maximum likelihood estimation (MLE) for the DM distribution. Building on the framework introduced by Sklar, we present a fast MLE implementation for the BLM and show the necessary conditions for its convergence during Newton-Raphson optimization. We then explore conditions under which Sklar's approach is computationally efficient compared to vectorized and approximate versions of the Newton-Raphson algorithm. Next, we perform a power analysis for the BLM distribution using simulated data. Finally, we evaluate the accuracy of the DM and BLM against gold standard datasets using a variety of classification techniques. Overall, we show that both the DM and BLM can be computationally efficient during MLE, and the DM and BLM MLEs outperform the multinomial for datasets with lower levels of class overlap, while the multinomial distribution demonstrates comparable or better accuracy otherwise.

4.3 Methods

4.3.1 Definitions

We provide the following definitions to simplify derivations for the Dirichlet Multinomial (DM) and Beta-Liouville Multinomial (BLM) distributions [108]. The gamma function of a strictly positive integer a is,

$$\Gamma(a) = (a-1)!$$

The dual-input gamma function for strictly positive integers a and b is,

$$\Gamma(a,b) = \frac{\Gamma(a+b)}{\Gamma(a)} = \prod_{i=0}^{b-1}(a+i)$$

The dual-input log-gamma function is the log of the dual-input gamma function,

$$\ln\Gamma(a,b) = \sum_{i=0}^{b-1}\ln(a+i)$$

We also define $\underset{N\times(D+1)}{\boldsymbol{X}}$ to be an observed data matrix of non-negative integers. $\boldsymbol{X}$ contains N observations of a $D+1$ dimensional data vector $\boldsymbol{x}$, where $D+1$ is the number of categories being counted (Table 4.1).

Table 4.1: Example of the data matrix $\boldsymbol{X}$ with N observations and categories $d = 1 \ldots D+1$

	Category			
Observation	$d=1$	$d=2$	$\ldots$	$D+1$
$n=1$	3	17	$\ldots$	8
$n=2$	0	23	$\ldots$	11
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
N	20	0	$\ldots$	9

4.3.2 Derivations for the beta-Liouville multinomial distribution and the associated likelihood function

Liouville distributions of the second kind have D parameters $(\alpha_1, \alpha_2, \ldots, \alpha_D)$ and additional parameters ξ associated with a generating density function $f(\cdot)$ [106],

$$P(\boldsymbol{p}|\alpha_1,\ldots,\alpha_D,\xi) = f(u|\xi)\frac{\Gamma(\sum_{d=1}^{D}\alpha_d)}{u^{\sum_{d=1}^{D}\alpha_d-1}}\prod_{d=1}^{D}\frac{p_d^{\alpha_d-1}}{\Gamma(\alpha_d)} \tag{4.1}$$

where $u = \sum_{d=1}^{D} p_d < 1, p_d > 0, d = 1 \ldots D$, and D as defined above. The Beta-Liouville distribution is formed when using the beta distribution with parameters $\xi = \{\alpha, \beta\}$ as the generating density function for u, resulting in,

$$P(\boldsymbol{p}|\boldsymbol{\theta}) = \frac{\Gamma(\sum_{d=1}^{D} \alpha_d)\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} u^{\alpha-\sum_{d=1}^{D}\alpha_d}(1-u)^{\beta-1}\prod_{d=1}^{D}\frac{p_d^{\alpha_d-1}}{\Gamma(\alpha_d)} \tag{4.2}$$

where $\boldsymbol{\theta} = (\alpha_1, \ldots, \alpha_D, \alpha, \beta)$. Unlike the Dirichlet distribution, which has $D+1$ parameters, equal to the number of observed categories, the Beta-Liouville distribution has $D+2$ parameters. Similar to the Dirichlet, the Beta-Liouville distribution presented in equation (4.2) can be used as a conjugate prior to the multinomial distribution resulting in the BLM distribution,

$$\begin{aligned} P(\boldsymbol{x}, \boldsymbol{p}|\boldsymbol{\theta}) = & \left[\frac{\Gamma(\sum_{d=1}^{D}\alpha_d)\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)\prod_{d=1}^{D}\Gamma(\alpha_d)}\left(\sum_{d=1}^{D} p_d\right)^{\alpha'-\sum_{d=1}^{D}\alpha'_d}\right] \\ & \times \left[\frac{\Gamma((\sum_{d=1}^{D+1} x_d)+1)}{\prod_{d=1}^{D+1}\Gamma(x_d+1)}\left(1-\sum_{d=1}^{D} p_d\right)^{\beta'-1}\prod_{d=1}^{D} p_d^{\alpha'_d-1}\right] \end{aligned} \tag{4.3}$$

where $\alpha'_d = \alpha_d + x_d, \alpha' = \alpha + \sum_{d=1}^{D} x_d$, and $\beta' = \beta + x_{D+1}$, following Bouguila's notation (Bouguila 2011). Marginalizing over $\boldsymbol{p}$ and assuming that the data vector $\boldsymbol{x}$ has been observed results in the likelihood function,

$$\begin{aligned} P(\boldsymbol{x}|\boldsymbol{\theta}) &= \int_{\boldsymbol{p}} P(\boldsymbol{x}, \boldsymbol{p}|\boldsymbol{\theta}) d\boldsymbol{p} \\ &= \frac{\Gamma\left(1+\sum_{d=1}^{D+1} x_d\right)\Gamma\left(\sum_{d=1}^{D}\alpha_d\right)\Gamma(\alpha+\beta)\Gamma\left(\alpha+\sum_{d=1}^{D} x_d\right)\Gamma(\beta+x_{D+1})\prod_{d=1}^{D}\Gamma(\alpha_d+x_d)}{\left(\prod_{d=1}^{D+1}\Gamma(x_d+1)\right)\Gamma\left(\sum_{d=1}^{D}(\alpha_d+x_d)\right)\Gamma\left(\alpha+\beta+\sum_{d=1}^{D+1} x_d\right)\Gamma(\alpha)\Gamma(\beta)\prod_{d=1}^{D}\Gamma(\alpha_d)} \end{aligned} \tag{4.4}$$

Utilizing (4.4) and the dual-input gamma function, introduced above, the log-likelihood function can be written as follows for the entire data matrix $\boldsymbol{X}$,

$$\begin{aligned}\ell(\boldsymbol{\theta}) &= \ln(P(\boldsymbol{X}|\boldsymbol{\theta})) \\ &= \sum_{n=1}^{N}\left[\ln\Gamma(1,\sum_{d=1}^{D+1}x_{nd}) + \ln\Gamma\left(\alpha,\sum_{d=1}^{D}x_{nd}\right) + \ln\Gamma(\beta,x_{n,D+1}) - \ln\Gamma\left(\sum_{d=1}^{D}\alpha_d,\sum_{d=1}^{D}x_{nd}\right)\right] \\ &\quad + \sum_{n=1}^{N}\left[\left(\sum_{d=1}^{D}\ln\Gamma(\alpha_d,x_{nd})\right) - \ln\Gamma(\alpha+\beta,\sum_{d=1}^{D+1}x_{nd}) - \left(\sum_{d=1}^{D+1}\ln\Gamma(1,x_{nd})\right)\right]\end{aligned} \tag{4.5}$$

Utilizing the definition of the dual-input log-gamma function and rearranging equation (4.5), removing constant terms that do not involve the parameters, we have,

$$\begin{aligned}\ell(\boldsymbol{\theta}) \sim &\sum_{n=1}^{N}\left[\sum_{d=1}^{D}\sum_{i=0}^{x_{nd}-1}\ln(\alpha_d+i) - \sum_{i=0}^{(\sum_D x_{nd})-1}\ln\left(\sum_{d=1}^{D}\alpha_d+i\right)\right] \\ &+ \sum_{n=1}^{N}\left[\sum_{i=0}^{(\sum_D x_{nd})-1}\ln(\alpha+i) + \sum_{i=0}^{(x_{n,D+1})-1}\ln(\beta+i) - \sum_{i=0}^{(\sum_{D+1} x_{nd})-1}\ln(\alpha+\beta+i)\right]\end{aligned} \tag{4.6}$$

We seek to find the maximum likelihood estimates (MLEs) of $\boldsymbol{\theta}$ that optimize (4.6),

$$MLE(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\text{argmax}}\ \ell(\boldsymbol{\theta})$$

using the following optimization approaches.

4.3.3 MLE of the Beta-Liouville Multinomial distribution

In what follows, we use the Newton-Raphson algorithm for optimization, with step as in the following equation, utilizing the log-likelihood function (4.6) to find the MLEs.

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} - \boldsymbol{H}^{-1}\boldsymbol{g} \tag{4.7}$$

where $\boldsymbol{g}$ is the gradient vector and $\boldsymbol{H}$ is the Hessian matrix. The Hessian matrix is block diagonal [109] and its inverse is represented as follows,

$$\boldsymbol{H}(\ell(\boldsymbol{\theta}))^{-1} = \text{block-diag}\left\{\boldsymbol{H}(\ell(\alpha_1,\ldots,\alpha_D))^{-1}, \boldsymbol{H}(\ell(\alpha,\beta))^{-1}\right\} \tag{4.8}$$

The gradient and Hessian of (4.6) are therefore,

$$g_{\alpha_d} = \sum_{n=1}^{N}\left[\sum_{i=0}^{x_{nd}-1}(\alpha_d+i)^{-1} - \sum_{i=0}^{(\sum_D x_{nd})-1}\left(\sum_{d=1}^{D}\alpha_d+i\right)^{-1}\right], \text{ for } d \in 1\ldots D$$

$$g_{\alpha} = \sum_{n=1}^{N}\left[\sum_{i=0}^{(\sum_D x_{nd})-1}(\alpha+i)^{-1} - \sum_{i=0}^{(\sum_{D+1} x_{nd})-1}(\alpha+\beta+i)^{-1}\right] \tag{4.9}$$

$$g_{\beta} = \sum_{n=1}^{N}\left[\sum_{i=0}^{x_{n(D+1)}-1}(\beta+i)^{-1} - \sum_{i=0}^{(\sum_{D+1} x_{nd})-1}(\alpha+\beta+i)^{-1}\right]$$

$$\boldsymbol{H}(\ell(\alpha_1,\ldots,\alpha_D)) = \sum_{n=1}^{N}\begin{bmatrix} c_{\alpha_d} - \sum_{i=0}^{x_{n1}-1}(\alpha_1+i)^{-2} & \ldots & c_{\alpha_d} \\ \vdots & \ddots & \vdots \\ c_{\alpha_d} & \ldots & c_{\alpha_d} - \sum_{i=0}^{x_{nD}-1}(\alpha_D+i)^{-2} \end{bmatrix} \tag{4.10}$$

$$\boldsymbol{H}(\ell(\alpha,\beta)) = \sum_{n=1}^{N} \begin{bmatrix} c_{\alpha\beta} - \sum_{i=0}^{(\sum_D x_{nd})-1} (\alpha+i)^{-2} & c_{\alpha\beta} \\ c_{\alpha\beta} & c_{\alpha\beta} - \sum_{i=0}^{x_{n(D+1)}-1} (\beta+i)^{-2} \end{bmatrix} \tag{4.11}$$

where the constants $c_{\alpha_d} = \sum_{i=0}^{(\sum_D x_{nd})-1} \left(\sum_{d=1}^{D} \alpha_d + i \right)^{-2}$ and $c_{\alpha\beta} = \sum_{i=0}^{(\sum_{D+1} x_{nd})-1} (\alpha+\beta+i)^{-2}$. The following formulae were used to invert the Hessian efficiently [110].

$$\boldsymbol{H} = \text{diag}(\boldsymbol{h}) + \mathbf{1}\mathbf{1}^{\top} c$$

$$\boldsymbol{H}^{-1} = \text{diag}(\boldsymbol{h}^{-1}) - \frac{\boldsymbol{h}^{-1}(\boldsymbol{h}^{-1})^{\top}}{c^{-1} + \sum_{d=1}^{D} h_d^{-1}} \tag{4.12}$$

where the diag$(\cdot)$ function places a given vector on the diagonal of a matrix of appropriate dimension, $\boldsymbol{h}$ is a column vector containing the non-constant terms from the diagonal of the Hessian, $\mathbf{1}$ is a column vector of ones of the appropriate dimension, and c is the constant term. The conditions for which this optimization process maintains convexity required for convergence can be found in Appendix 1. Using (4.7) to perform Newton-Raphson steps during parameter estimation, we considered several computational methods for calculating the gradient and Hessian to compare their accuracy and runtime performance.

Vectorized MLE

We first explored the standard approach of exactly calculating the above gradient and Hessian equations using Python Numpy's vectorized operations to improve computational efficiency. This includes Single-Instruction Multiple Data (SIMD) operations using Streaming SIMD Extensions 2 (SSE2) and intrinsic optimizations of data structures for vector and matrix operations.

Approximate MLE

Next, we examined finite limit approximations of the repetitive sums in (4.6) to avoid the computational cost associated with an increasing number of parameters, observations, and data draws. For this approach, we attempted to approximate the gradient and Hessian finite sums by fitting functions to generalized formulae of this structure:

$$f(\theta, N) = \sum_{n=0}^{N} \frac{1}{\theta + n} = \frac{1}{\theta} + \sum_{n=1}^{N} \frac{1}{\theta + n} \tag{4.13}$$

$$f(\theta, N) = \sum_{n=0}^{N} \frac{1}{(\theta + n)^2} = \frac{1}{\theta^2} + \sum_{n=1}^{N} \frac{1}{(\theta + n)^2} \tag{4.14}$$

for positive real θ and non-negative integer N. These formulae, ignoring the term where $n = 0$, are extensions of the finite limits for the Harmonic and Geometric series, respectively. Finite approximations of the divergent Harmonic series and convergent Geometric series have been extensively studied [111, 112]. However, approximations of these series that include a second parameter in the denominator have, to our knowledge, not been published.

Starting with the known approximation of the trivial case where $\theta \to 0$, manual function fitting was performed by trial and error using the R function "nls" for assistance. The resulting approximations were then used to compute the finite sums in (4.6) for the Newton-Raphson procedure. Detailed explanations of the limit approximations are available in Supplementary File 1.

Sklar's MLE

Sklar (2014) outlines an approach to reduce the computational cost of the repetitive sums in (4.6) by introducing new data structures. Here, we extend his approach to the BLM distribution and compare it to the vectorized and approximate MLE approaches.

As noted above, to compute (4.6) naively would involve:

- For terms involving α_d and β, which depend on the count at position n, d, repeating sums for each category d less than that of the max value in row n of the data matrix.

- For terms involving $\sum_D \alpha_d$, α, and $\alpha + \beta$, which depend on the partial or complete row sum of row n, repeating sums for each row less than that of the max row sum.

To circumvent this, Sklar creates two data structures to hold the multiplicative constants to replace these repetitive sums. Here, due to the increased complexity of the BLM distribution, we need three such data structures. First, we define Z_D, Z_{D+1}, and $Z_{\max}$ as follows,

$$Z_D = \max_n \sum_{d=1}^{D} x_{nd}$$
$$Z_{D+1} = \max_n \sum_{d=1}^{D+1} x_{nd}$$
$$Z_{\max} = \max\left(Z_D, Z_{D+1}\right)$$

The first data structure is a matrix $\underset{(D+1)\times Z_{\max}}{\boldsymbol{U}}$, which, for each category $d \in 1 \ldots D+1$, counts the number of rows where the count in column d of $\boldsymbol{X}$ exceeds the integer z, where z is the zero-based column index of $\boldsymbol{U}$, $z = 0 \ldots Z_{\max} - 1$.

The second data structure is a vector $\boldsymbol{v}^D$ of length Z_D, which, for each category $d \in 1 \ldots D$, counts the number of rows where the row sum exceeds z, where z is the zero-based index of $\boldsymbol{v}$, $z = 0 \ldots Z_D - 1$.

The third data structure is another vector $\boldsymbol{v}^{D+1}$ of length Z_{D+1}, which, for each category $d \in 1 \ldots D+1$, counts the number of rows where the row sum exceeds z, where z is the zero-based index of $\boldsymbol{v}^{D+1}$, $z = 0 \ldots Z_{D+1} - 1$.

Note that the dimensionality of the matrix $\boldsymbol{U}$ is such that many of its values on the right hand side will be zero using this definition; this is for notational convenience in the derivations. In practice, the data structure for $\boldsymbol{U}$ can be stored efficiently as a ragged array containing only non-zero elements.

More formally:

$$
\begin{aligned}
u_{dz} &= \sum_{n=1}^{N} \mathbb{I}\left(x_{nd} > z\right) \\
v_z^D &= \sum_{n=1}^{N} \mathbb{I}\left(\sum_{d=1}^{D} x_{nd} > z\right) \\
v_z^{D+1} &= \sum_{n=1}^{N} \mathbb{I}\left(\sum_{d=1}^{D+1} x_{nd} > z\right)
\end{aligned}
$$

Where $\mathbb{I}$ is the indicator function:

$$
\mathbb{I}(\cdot) \coloneqq \begin{cases} 1 & \text{if true} \\ 0 & \text{otherwise} \end{cases}
$$

This simplifies (4.6):

$$
\begin{aligned}
\ell(\boldsymbol{\theta}) \sim \sum_{z=0}^{Z_D-1} \left[\sum_{d=1}^{D} u_{dz} \ln(\alpha_d + z) + v_z^D \ln(\alpha + z) - v_z^D \ln(\sum_{d=1}^{D} \alpha_d + z)\right] \\
+ \sum_{z=0}^{Z_{D+1}-1} \left[u_{(D+1)z} \ln(\beta + z) - v_z^{D+1} \ln(\alpha + \beta + z)\right]
\end{aligned} \tag{4.15}
$$

The gradient and Hessian of (16) are then,

$$
g_{\alpha_d} = \sum_{z=0}^{Z_D-1} \left[u_{dz}(\alpha_d + z)^{-1} - v_z^D(\sum_{d=1}^{D} \alpha_d + z)^{-1}\right]
$$

$$
g_\alpha = \sum_{z=0}^{Z_D-1} v_z^D(\alpha + z)^{-1} - \sum_{z=0}^{Z_{D+1}-1} v_z^{D+1}(\alpha + \beta + z)^{-1} \tag{4.16}
$$

$$g_\beta = \sum_{z=0}^{Z_{D+1}-1} \left[u_{(D+1)z}(\beta + z)^{-1} - v_z^{D+1}(\alpha + \beta + z)^{-1} \right]$$

$$\boldsymbol{H}(\ell(\alpha_1, \ldots, \alpha_D)) = \sum_{z=0}^{Z_D-1} \begin{bmatrix} c_{\alpha_d} - u_{dz}(\alpha_d + z)^{-2} & \ldots & c_{\alpha_d} \\ \vdots & \ddots & \vdots \\ c_{\alpha_d} & \ldots & c_{\alpha_d} - u_{dz}(\alpha_d + z)^{-2} \end{bmatrix} \tag{4.17}$$

$$\boldsymbol{H}(\ell(\alpha, \beta)) = \sum_{z=0}^{Z_{D+1}-1} \begin{bmatrix} c_{\alpha\beta} - \sum_{z=0}^{Z_D-1} v_z^D(\alpha + z)^{-2} & c_{\alpha\beta} \\ c_{\alpha\beta} & c_{\alpha\beta} - u_{(D+1)z}(\beta + z)^{-2} \end{bmatrix} \tag{4.18}$$

where the constants $c_{\alpha_d} = v_z^D(\sum_{d=1}^D \alpha_d + z)^{-2}$ and $c_{\alpha\beta} = v_z^{D+1}(\alpha + \beta + z)^{-2}$.

4.3.4 MLE runtime analysis

Theoretical and empirical runtimes for the MLE Newton-Raphson procedure were assessed for the DM and BLM distributions using the three computational methods listed above: vectorized, approximate, and Sklar. For empirical runtime assessment, the number of observations N, the number of data draws from each observation vector $M = \sum_{D+1} x_d$, and the number of categories $D + 1$, were each varied while holding the others constant. Wall-clock time for a single Newton-Raphson MLE step was then computed for each of the above algorithms for the DM and BLM distributions. Five independent bootstraps were performed for each experiment to account for variance due to hardware performance. All runtime experiments were performed on Ubuntu Linux 18.04 using Python 3.7.3, Numpy 1.18.1, an 18-core Intel i9-9980XE Skylake 3.0 GHz processor, 64 GB DDR4 3600 SDRAM, and an M.2 2280 NVMe 1.3 V-NAND solid state drive.

4.3.5 Classification performance benchmarking

We aimed to characterize the effectiveness of the BLM distribution for classifying labelled data, commonly known as supervised classification. In the following sections, we compare the BLM to the DM and multinomial distributions using both simulated data and real-life, gold standard datasets, all of which have known class labels. Simulated data allows for fine control over the difficulty of the classification task and demonstrates for which datasets these distributions are effective. Gold standard datasets offer additional value over simulated data, as they often contain noise unrelated to the classification task and are therefore more representative of how each distribution will perform in general.

In the following experiments, we compare standard multinomial naive Bayes and two MLE-based strategies for performing the classification task. Each of these strategies uses different information from the training and test data, however all methods involve maximizing the likelihood function in some way. For the standard multinomial and the first of the MLE-based classification strategies, we construct a test matrix of counts for each feature $\underset{N\times(D+1)}{\boldsymbol{T}}$ and a training matrix with columns representing each class, $c \in C$, on the unit simplex $\underset{(D+1)\times C}{\boldsymbol{X}}$. Classification of multiple test observations is then performed by computing the matrix inner product and taking the argmax for each row in the resulting matrix, which corresponds to the assigned class for each test observation. A formal explanation of this process is available in Supplementary File 1. The second MLE-based approach involves explicit calculation of the marginal likelihood for each class and does not involve calculating the matrix inner product. Below, we provide additional details about these methods, referred to here as multinomial naive Bayes, Dirichlet and Beta-Liouville multinomial naive Bayes, and marginal likelihood classification, respectively.

Multinomial naive Bayes

Parameters on the unit simplex are determined by simple division for multinomial naive Bayes. In this approach, observations $n \in N$ within the training data are summed to produce a single count vector, and the counts are divided by the vector sum to produce the training parameters,

equivalent to the MLEs of the multinomial, $\hat{\boldsymbol{p}}$. The resulting trained parameters are then used to classify test observations by matrix inner product.

$$\hat{p}_d = \frac{\sum_N x_{nd}}{\sum_N \sum_{D+1} x_{nd}}, \text{ for } d = 1 \dots D+1 \tag{4.19}$$

Dirichlet and beta-Liouville multinomial naive Bayes

In this approach, the training data are used to perform Newton-Raphson MLE separately for each class, and the resulting $D+1$ and $D+2$ MLEs are used to calculate the $D+1$ training parameters for the DM and BLM distributions, respectively. The resulting trained parameters are then used to classify test observations by matrix inner product.

DM Distribution:

$$\hat{p}_d = \frac{\hat{\alpha}_d}{\sum_{D+1} \hat{\alpha}_d}, \text{ for } d = 1 \dots D+1 \tag{4.20}$$

BLM Distribution:

$$\hat{p}_d = \frac{\hat{\alpha}}{\hat{\alpha}+\hat{\beta}} \frac{\hat{\alpha}_d}{\sum_{D+1} \hat{\alpha}_d}, \text{ for } d = 1 \dots D \tag{4.21}$$

$$\hat{p}_{D+1} = \frac{\hat{\beta}}{\hat{\alpha}+\hat{\beta}}$$

Where $\hat{\alpha}_d$, $\hat{\alpha}$, and $\hat{\beta}$ are the MLEs.

Marginal likelihood classification

Like the previous approach, the training data are used to calculate MLE parameter estimates for $D+1$ and $D+2$ parameters for the DM and BLM distributions, separately for each class. However, for classification, the marginal likelihood function $P(\boldsymbol{x}|\boldsymbol{\theta})$ is evaluated separately for each test observation vector $\boldsymbol{x}$ and class parameter MLE estimates $\boldsymbol{\theta}_c, c \in C$. The test observation is then assigned to the class with the maximum likelihood.

DM Distribution:

$$\begin{aligned}\ln(P(\boldsymbol{x}|\boldsymbol{\theta}_c)) = &\sum_{i=0}^{(\sum_{D+1} x_d)-1} \ln(1+i) - \sum_{i=0}^{x_d-1} \ln(1+i) \\ &+ \sum_{d=1}^{D+1} \sum_{i=0}^{x_d-1} \ln(\hat{\alpha}_d + i) - \sum_{i=0}^{(\sum_{D+1} x_d)-1} \ln(\sum_{d=1}^{D+1} \hat{\alpha}_d + i)\end{aligned} \tag{4.22}$$

BLM Distribution:

$$\begin{aligned}\ln(P(\boldsymbol{x}|\boldsymbol{\theta}_c)) = &\sum_{i=0}^{(\sum_{D+1} x_d)-1} \ln(1+i) - \sum_{d=1}^{D+1} \sum_{i=0}^{x_d-1} \ln(1+i) \\ &+ \sum_{d=1}^{D} \sum_{i=0}^{x_d-1} \ln(\hat{\alpha}_d + i) - \sum_{i=0}^{(\sum_{D} x_d)-1} \ln\left(\sum_{d=1}^{D} \hat{\alpha}_d + i\right) \\ + &\sum_{i=0}^{(\sum_{D} x_d)-1} \ln(\hat{\alpha} + i) + \sum_{i=0}^{(x_{D+1})-1} \ln(\hat{\beta} + i) - \sum_{i=0}^{\left(\sum_{D+1} x_d\right)-1} \ln(\hat{\alpha} + \hat{\beta} + i)\end{aligned} \tag{4.23}$$

Finally, Laplace additive smoothing was applied to all of the above methods such that all categories had non-zero training parameter probability even if they were not observed in the training data [88, 103, 113].

Classification power analysis on simulated data

A power analysis was performed for the DM and BLM distributions to assess and compare classification accuracy on simulated data. Data were simulated to achieve different levels of overlap and sampling for 4 distinct classes. Benchmarking on these data involved assessing the accuracy of classification back to the known class labels using standard performance measures, including precision, recall, specificity, and F1 score. Data were generated as follows (Figure 4.1).

- We set the number of categories, $D + 1$, within each class to be 100. These are categories of a generating multinomial with parameters $\boldsymbol{p} = \{p_1, \ldots, p_{100}\}$.

- We chose fixed, equidistant categories to center each class: category 1 for class 1, category 33 for class 2, category 66 for class 3, and category 100 for class 4. These centers represent the probability density mode of the associated multinomial for each class (Figure 4.1a).

- The vector $\boldsymbol{p}$ was generated as follows for each class: 1) We used the central category of the class as a mean of a normal distribution and varied the standard deviation to create more or less overlap between class probability densities (Figure 4.1b). Five standard deviation values were explored: 10, 22, 35, 48, and 60. 2) We randomly sampled 2000 draws from these normal distributions associated with each class. For example, sampling from the normal distribution with mean 33 and standard deviation 10 associated with class 2 results in data points spread, for the most part, between category 16 and category 49 with a mode around the 33 mean. We created an empirical distribution by rounding the sampled numbers to integers and counting the integers with the same value. For example, we could count the number of integers that were equal to 30 (category 30). 3) For each class we divided the totals per category by the total number of draws to construct the probability vector $\boldsymbol{p}$ of the generating multinomials. Large standard deviation values result in more overlap between the generating multinomials per class, while small standard deviations result in more class separation with concentrated probability density around the mean.

- The resulting multinomials with different degree of overlap were used to generate count data for subsequent classification. To assess the impact of sampling on these results we varied the total number, M, of samples (data draws) obtained from the generating multinomials. Values of 15, 136, 258, 379, and 500 were explored for M. Small values of M result in sparse simulated count vectors, while large values of M result in dense count vectors that better inform the maximum likelihood estimation process.

- Utilizing the same generating multinomial distributions, we also varied the number of count vectors N (data observations) generated for each class. Values of 2, 26, 51, 76, and 100 were

explored for N. Larger values of N should result in better parameter estimates and therefore more accurate classification.

These datasets were preprocessed as outlined in Supplementary File 1 and classified using the methods introduced in previous sections.

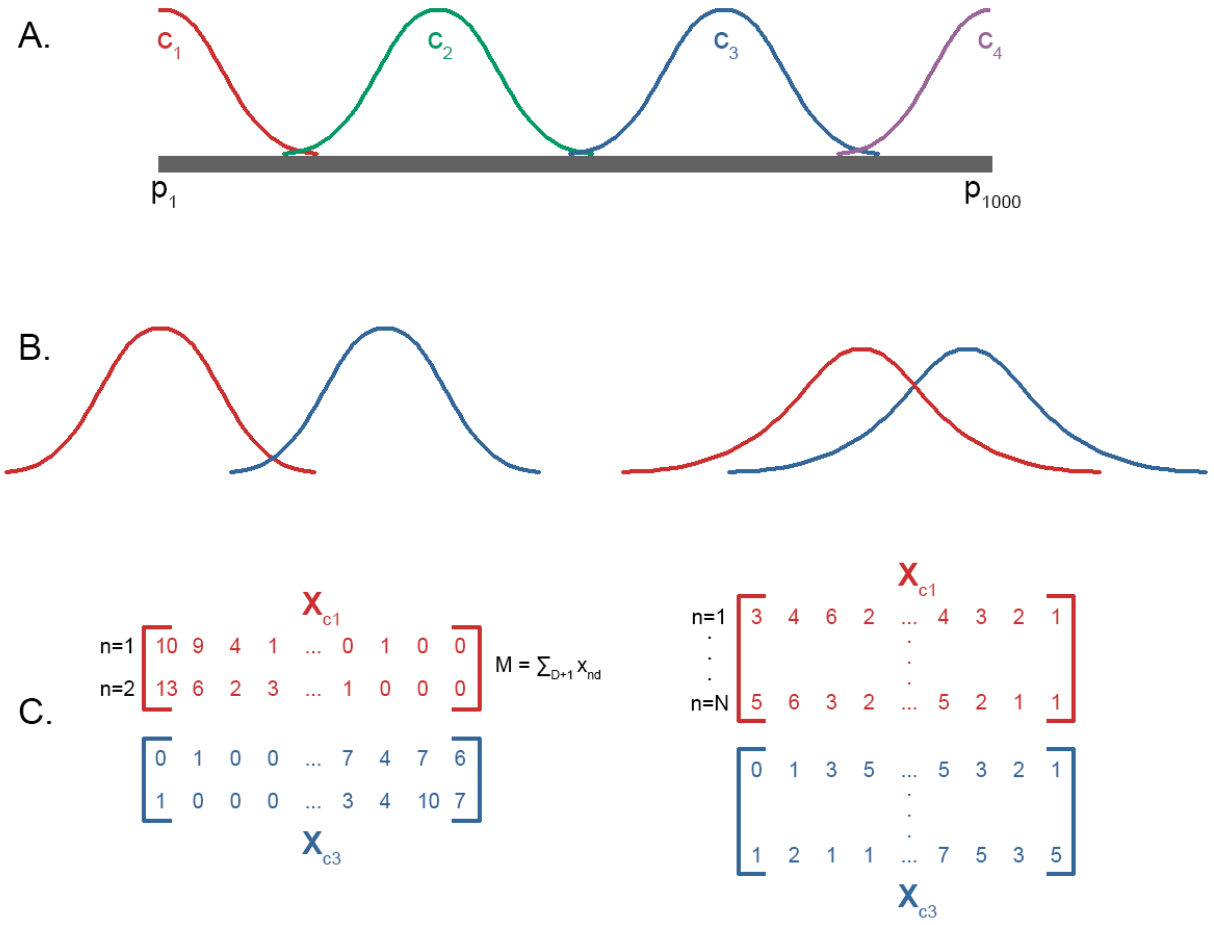

Figure 4.1: Data were simulated for 4 discrete classes according to normal distributions evenly spaced along a vector of 100 categories. A) Each class had normal probability density with mean equal to a category varying equidistantly from p_1 to p_{100}. B) For each class, the standard deviation of the normal distribution was varied, $\sigma = \{10, 22, 35, 48, 60\}$, with higher values creating more overlap between classes. C) Using the resulting multinomial parameters, $M = \{15, 136, 258, 379, 500\}$ counts were randomly sampled for $N = \{2, 26, 51, 76, 100\}$ independent data vectors for each class. These counts were then classified according to previously stated methods and used to calculate accuracy measures.

Finally, we aimed to determine if the BLM distribution performed better on count data derived from a BLM distribution as opposed to multinomial data. To achieve this, multinomial generating parameters were simulated as above and were used in a Markov-chain Monte Carlo (MCMC) process to produce BLM data with target distribution,

$$P(\boldsymbol{x}_c|\boldsymbol{\theta}_c) = \frac{P(\boldsymbol{\theta}_c|\boldsymbol{x}_c)P(\boldsymbol{x}_c)}{P(\boldsymbol{\theta}_c)} \tag{4.24}$$

where $\boldsymbol{x_c}$ is a count data vector, and $\boldsymbol{\theta_c}$ are the fixed, generating parameters, for class c. A formal description of the MCMC data simulation process can be found in Supplementary File 1. MCMC simulation was performed separately for each class using a Metropolis-Hastings sampling strategy with a multinomial proposal, $P(\boldsymbol{x})$, and acceptance-rejection probability,

$$\alpha = \min\left\{1, \frac{P(\boldsymbol{x}_{i+1}|\boldsymbol{\theta}_c)/P(\boldsymbol{x}_{i+1})}{P(\boldsymbol{x}_i|\boldsymbol{\theta}_c)/P(\boldsymbol{x}_i)}\right\} \tag{4.25}$$

where $P(\boldsymbol{x}_{i+1}|\boldsymbol{\theta}_c)$ is the likelihood of $\boldsymbol{x}_{i+1}$ under the BLM and $P(\boldsymbol{x}_{i+1})$ is the likelihood of the data under the proposal. The data generation process for both multinomial and BLM data was independently replicated 5 times for each set of variables, and the resulting datasets were classified using the same procedures introduced in the previous sections.

Classification on Gold Standard Datasets

Performance on real datasets was determined using four gold standard datasets from the text-based natural language processing domain [114] described below:

- **20 Newsgroups**: an English text collection of 18,821 newsgroup documents categorized into 20 classes with a relatively even class distribution. The dataset used in this experiment was split within each class into 11,293 training and 7,528 testing documents.
- **Reuters-21578 (R8)**: an English text collection of 7,674 newswire documents from the Reuters Ltd. group in 1987. Only the 10 most frequent classes were used for this dataset,

and only observations belonging to a single, unique class label were included. This resulted in 8 classes with more than one document, hence the name R8. This dataset has a heavily skewed class distribution and was split within each class into 5,485 training and 2,189 test documents.

- **Cade12**: a Brazilian Portuguese text collection of 40,983 webpages from the CADE web directory. Documents are categorized into 12 classes with a skewed class distribution. This dataset was split within each class into 27,322 training and 13,661 test documents.

- **WebKB**: an English text collection of 4,199 university webpages from computer science departments participating in the World Wide Knowledge Base project in 1997. Documents are categorized into 4 classes with a moderately skewed class distribution. This dataset was split within each class into 2,803 training and 1,396 test documents.

4.4 Results

MLE runtimes for the BLM distribution were comparable to the DM distribution but with a small increase in runtime due to computation of additional terms in the likelihood function and its derivatives (Figure 4.2). Performance of the BLM distribution on simulated data was nearly identical to the DM distribution's performance. However, the BLM distribution was not as accurate on very sparse data with few observations, N, and few data draws, M, suggesting that the BLM distribution requires more data density than the DM to produce accurate MLEs. Finally, the multinomial distribution performed most consistently on gold standard data and performed best on two of the four datasets examined. However, the DM and BLM distributions were able to outperform the multinomial in accuracy by a small margin on the other datasets. Additional details about the results of each experiment are provided below.

4.4.1 MLE runtime results

Overall, the vectorized method was the most computationally efficient, particularly on data with a large number of parameters and dense vectors (a large number of data draws). The ap-

proximate method performed optimally for high values of data draws and low values of data observations. Sklar's method performed optimally with a large number of data observations and a small number of parameters and data draws; however, the runtime of Sklar's approach scaled poorly relative to the vectorized and approximate methods as the number of parameters increased. Of particular note are the middle and right panels in Figure 4.2, highlighting impact of an increased number of parameters and number of draws, respectively, on runtime. Figure 4.2 shows that Sklar's method scaled exponentially as parameter count increased, reaching almost 60 seconds per Newton-Raphson step at a moderate parameter count of 5,000. In contrast, the vectorized method scaled linearly, requiring only seconds per step, and the approximate method scaled nearly log-linearly with the number of data draws. This suggests that the vectorized and approximate approaches will perform most efficiently for a majority of real-world problems, for which a large number of parameters must be estimated.

4.4.2 Classification results

Overall, the standard multinomial naive Bayes classifier performed well on both simulated and gold standard data classification tasks. The DM and BLM MLEs resulted in marginally better classification performance for the 20 Newsgroups and WebKB gold standard datasets but were not as consistent as the multinomial classifier, likely due to the relative difficulty of parameter optimization compared to the simple division used to produce the multinomial parameter estimates. As expected, all distributions and methods improved in classification performance when more data were available, including both the number of observations N and the number of data draws M within each observation. The following sections explore more precisely how classification performance changed with varying amounts of data and how well these power estimates generalized to real-world data.

Simulated data power analysis results

The following variables affected classification power, ordered from most to least important: 1) difficulty of the classification task (i.e. class distribution overlap), 2) the density of the data

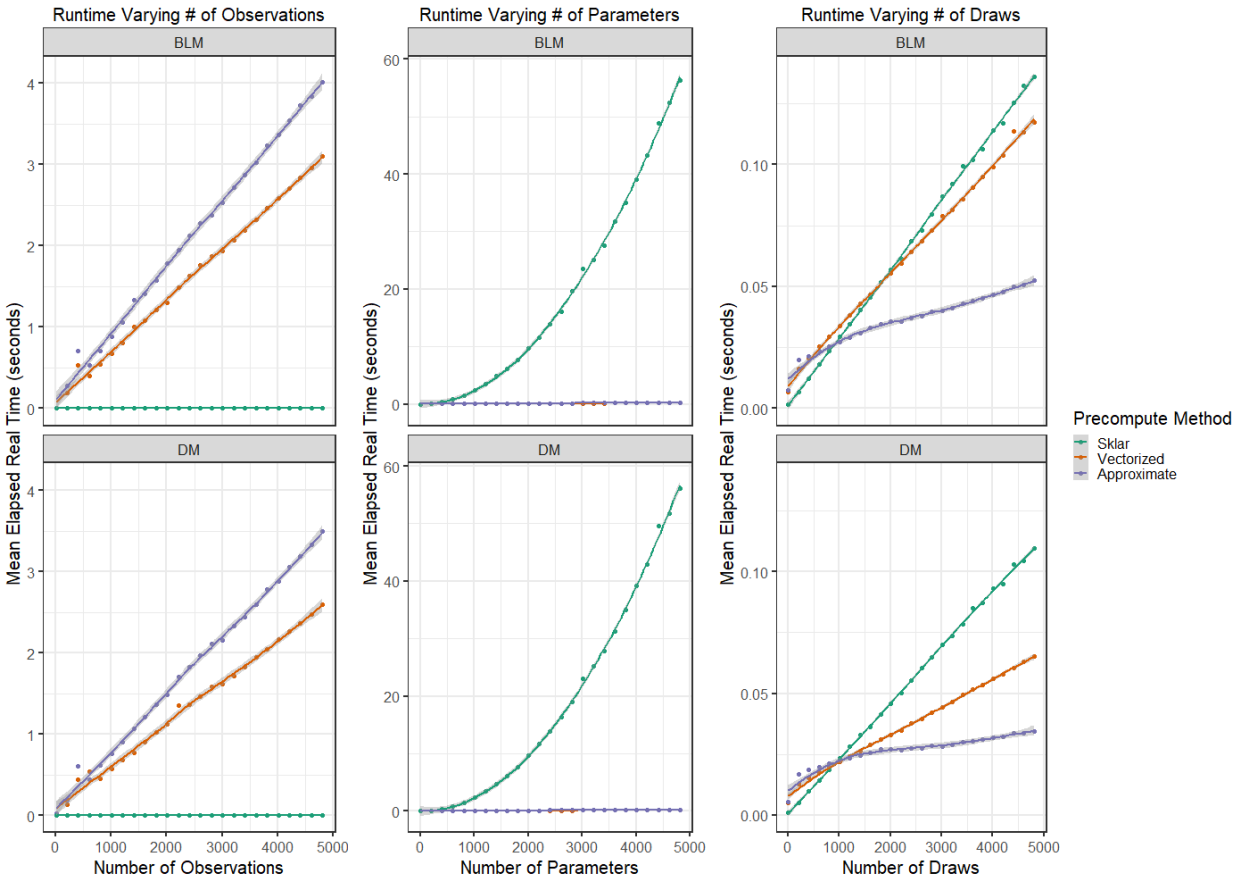

Figure 4.2: Vectorized and approximate methods scale more efficiently than Sklar's method with a large number of parameters and data draws. Sklar's approach scales more efficiently than the vectorized and approximate methods when the number of observations is large and the number of parameters and data draws are small. The approximate method is the most computationally efficient when the number of data draws is large, as it is capable of scaling log-linearly. In the above figures, each data point is the mean of 5 independent bootstraps, and a locally estimated scatterplot smoothing (LOESS) line, as implemented in the R ggplot2 package, is shown for each method [115]. Variance around the LOESS line is shown in gray.

vector (data draws), and 3) the number of data observations (Figure 4.3). Even for heavily overlapping distributions, median F1 score exceeded 90% for the matrix multiplication methods using the multinomial, DM, and BLM distributions at 26 observations and 136 data draws (Supplementary Figure 4.4 and Figure 4.5). The classification scores for classes 1 and 4 were typically higher than those for classes 2 and 3; this was due to classes 1 and 4 having less overlap than classes 2 and 3, resulting in an easier classification task (Figure 4.1). However, these class differences were resolved with additional training data, particularly for the datasets with less class overlap.

The marginal likelihood classification method required more data observations and data draws to achieve a similar F1 score on the same data, with the BLM distribution requiring more data than the DM distribution (Supplementary Figure 4.6 and Figure 4.7). Finally, data generated from the multinomial distribution did not appear to affect the F1 score when compared to data generated from the BLM distribution for all classification methods and distributions examined (Supplementary Figure 4.8 and Figure 4.9). This suggests that the major differences in classification power were due to differences in the distributions and classification methods but not the statistical origin of the data. Overall, these results demonstrate that the multinomial, DM, and BLM distributions are capable of achieving nearly perfect classification results under ideal conditions, provided enough training data are available.

Gold standard data results

Each of the DM, BLM, and multinomial distributions had the highest accuracy and F1 scores on two of the four gold standard datasets (Table 4.2). For all datasets and distributions, the naive Bayes classification methods using the matrix inner product produced the highest performance metrics. For the DM and BLM MLEs, the approximate compute method resulted in marginally better performance metrics than the exact methods (vectorized and Sklar) for the 20 Newsgroups and R8 datasets. However, the approximate method was not consistent and resulted in markedly lower accuracy and F1 scores for the Cade12 dataset. The lower accuracy of the approximate method on the Cade12 dataset could be due to the MLEs converging on large parameter values, for which the finite sum approximations diverge from exact values. This limitation of the approximate

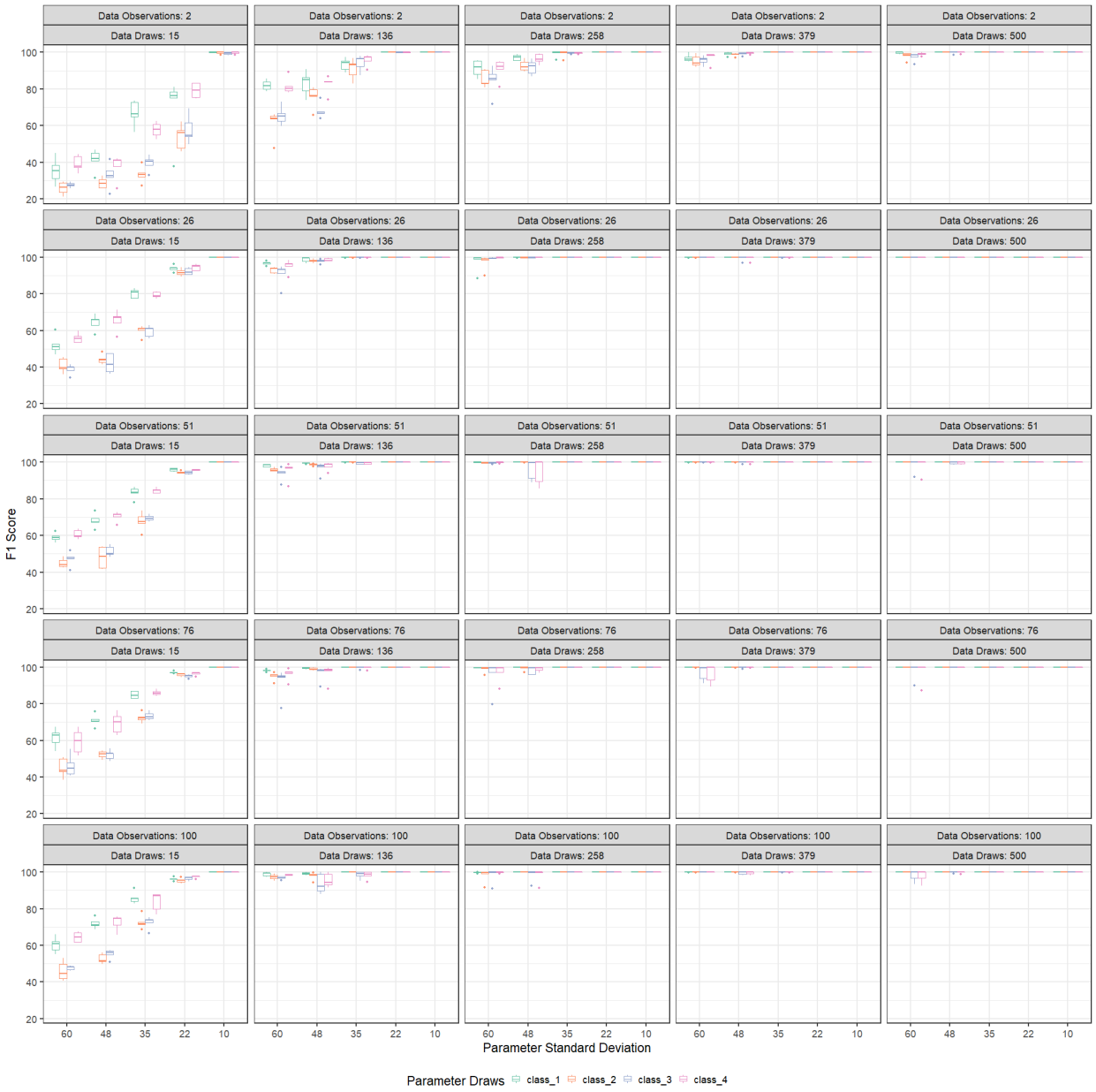

Figure 4.3: Use of the BLM distribution results in high F1 classification scores as data draws and observations increase, even for difficult classification tasks. Parameter standard deviation is a measure of classification difficulty: higher values result in more overlapping class distributions. Data draws (the density of the count vectors) has a greater affect on F1 score than data observations, however higher values of both result in higher F1 scores. A separate boxplot is shown for each simulated class distribution (classes 1-4). The boxplots summarize the F1 score quartiles for 5 independent datasets for each combination of variables.

method is discussed further in the following section. The multinomial distribution with a standard naive Bayes classifier performed consistently well across all datasets, while the DM and BLM MLEs were less consistent overall.

Of particular interest, the Cade12 dataset had markedly lower F1 scores (around 7% lower) for the MLEs than the multinomial distribution across all classification and compute methods. In contrast, the MLEs performed almost as well or better than the multinomial distribution for the remaining datasets to within nearly 1% F1 score. This suggests that the Cade12 dataset poses challenges for the accurate calculation of the MLEs, and it is well-known that the Cade12 data are more complex and difficult to classify than the other datasets evaluated here [116]. Additionally, the Cade12 dataset was the only dataset for which the approximate compute method and marginal likelihood classifiers produced extremely low F1 scores relative to the exact compute methods (vectorized/Sklar) using the naive Bayes classifier. These results are suggestive of numerical overflow during computation or that the Newton-Raphson method is traversing a difficult optimization landscape. In the discussion of these results, we outline additional criteria for which the approximate method breaks down that could have contributed to the low F1 scores.

Finally, we note that because classification results were calculated across all classes within each dataset, the F1 score was identical to both precision and recall; therefore precision and recall are not shown.

4.5 Discussion

Our results show that efficient methods exist for computation of MLEs for the Dirichlet Multinomial (DM) and Beta-Liouville Multinomial (BLM) distributions. For datasets where the number of categories (features $D+1$) to be estimated is small but the data matrix (observations N and count density M) is large, Sklar's computational approach would be the most efficient. However, it is common in natural language processing classification tasks to have many features (often more than 1,000) and sparse data available for MLE computation. In this more common case, the vec-

Table 4.2: The DM, BLM, and multinomial distributions each performed best on two of the four gold standard datasets. The best-performing combination of distribution and method are shown in bold for each dataset. In all datasets, the standard multinomial naive Bayes classifier performed consistently well, while the MLE-based methods were less consistent across datasets. The approximate compute method appeared to result in better MLEs for the 20 Newsgroups and R8 datasets, perhaps by preventing computational overflow. Percent accuracy and F1 score are presented for each combination of methods and datasets examined. Abbreviations: Dirichlet multinomial (DM), beta-Liouville multinomial (BLM), multinomial (multinom.) naive Bayes (NB), vectorized (vec), distribution (distr.), 20 Newsgroups (20 NG), accuracy (acc.).

Distr.	**Classifier**	**Method**	**20 NG**		**Reuters (R8)**		**Cade12**		**WebKB**	
			Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1
DM	NB	Vec/Sklar	98.2	82.2	98.7	94.8	91.7	50.3	93.0	86.0
		Approx.	**98.3**	**82.9**	**99.2**	**96.7**	86.6	19.6	93.1	86.1
	Marginal	Vec/Sklar	97.8	78.5	97.9	91.4	88.8	32.9	92.9	85.8
		Approx.	98.2	81.6	98.8	95.2	86.1	16.8	92.9	85.8
BLM	NB	Vec/Sklar	98.2	82.2	98.7	94.8	91.7	50.3	93.0	86.0
		Approx.	97.8	77.7	**99.2**	**96.7**	86.6	19.7	92.2	84.4
	Marginal	Vec/Sklar	97.8	78.1	97.7	90.6	86.4	18.6	**93.2**	**86.3**
		Approx.	97.7	76.6	98.8	95.2	85.8	15.0	90.4	80.7
Multinom.	NB	Multinom.	98.2	82.5	**99.2**	**96.7**	**93.0**	**57.9**	92.0	84.1

torized computation and the limit approximation approaches explored here would be far superior for computational efficiency.

Our results also indicate that the BLM distribution seems to perform equally as well as the DM distribution on both simulated and gold standard data, outperforming the DM distribution for certain datasets like WebKB (Figure 4.3, Table 4.2). Particularly, the BLM distribution performs well when sufficient data are present for optimization of the BLM parameters and when the class densities are mildly to moderately non-overlapping. For datasets where the class distributions are heavily overlapping and data are sparse, such as in the Cade12 dataset, the DM and BLM MLEs performed worse than the naive estimates produced by the standard multinomial naive Bayes procedure. This suggests that the optimization landscape for such datasets is not sufficiently easy to navigate for the Newton-Raphson procedure, resulting in poor MLEs. However, we note that the BLM and DM distributions may perform better under different circumstances like unsupervised classification, where class labels are not fixed, as has been shown by Bouguila 2008. The results observed here for the Cade12 dataset may be a limitation of the DM and BLM distributions in strictly supervised classification tasks.

An interesting result was the marginal gain in performance using the finite limit approximations, leading to a 0.7% F1 gain for the DM naive Bayes on the 20 Newsgroups dataset compared to all other methods and a 1.9% F1 gain for the BLM and DM naive Bayes on the R8 dataset compared to the MLE naive Bayes methods. This suggests that the approximated values produced a more rapidly-converging Newton-Raphson optimization landscape than the exact methods (vectorized/Sklar). It is also possible that the approximations helped to avoid numerical overflow in cases where $\theta \to 0$, which results in large values for the first iteration of the finite sums. Additionally, our approximations seemed to be accurate for small parameter values (θ); the parameter MLEs tended toward smaller values for the BLM and DM distributions on the 20 Newsgroups, R8, and WebKB datasets (data not shown). However, our approximations appeared to be inaccurate as parameter MLEs reached large values ($\theta > 1000$), as was reflected in the poor performance metrics for the DM and BLM distributions on the Cade12 dataset. If more exact approximations were found for these finite sums, they could result in increased computational efficiency and more rapid convergence for Newton-Raphson MLEs. Therefore, while our approximations produced an interesting result, further work must be done to derive more exact equations before they are employed for general use.

Though the BLM MLEs were successful on two of the four gold standard datasets, it became clear during our research that the BLM distribution has several limitations during Newton-Raphson optimization that must be avoided for its successful use in this context. We observed two types of pathological behavior during parameter optimization that hadn't been previously described, to the best of our knowledge. First, the Hessian matrix of the BLM requires that values for α_1, α, β meet certain conditions for the matrix to be negative definite (Appendix I). If the Hessian is not negative definite, then convergence of the Newton-Raphson procedure is not guaranteed. We solved this problem by checking these conditions before each Newton-Raphson step and adding small values to pathological parameters, such that the Hessian matrix was guaranteed to be negative definite. Second, even when convergence was guaranteed, we encountered certain datasets for which α, β had rapidly increasing parameter values, resulting in eventual numerical overflow. In these cases,

the ratio of $\frac{\beta}{\alpha}$ remained constant while the parameter estimates increased, suggesting the existence of local ridge optima in the landscape. This could be due to overparameterization of these particular datasets, suggesting that only $D + 1$ parameters are needed to model the data, while the BLM has $D + 2$ parameters. In these cases, it is recommended that the DM distribution be used instead of the BLM. The code accompanying this work contains examples of checking for these pathological behaviors and potential solutions to ensure convergent and accurate MLEs for the BLM distribution.

We acknowledge that the scope of this work is limited to naive Bayes classifiers, and that the power and gold standard data analyses presented here are not comprehensive. There may be other classifiers and datasets that demonstrate improved results for the DM and BLM distributions. Likewise, there may be datasets for which the DM and BLM distributions produce poor results regardless of the classification method used. Finally, while the DM and BLM distributions may produce marginal increases in accuracy for certain datasets, the time cost for parameter optimization may outweigh any increase in accuracy provided by their use. Given that the multinomial naive Bayes classifier performed consistently well and required a fraction of the computational cost, it may be the superior choice for supervised classification tasks.

Future work might explore additional optimization methods other than Newton-Raphson, additional classifiers other than naive Bayes, and broader classification tasks like unsupervised or streaming classification. Likewise, further exploration of finite sum approximations for the modified geometric and harmonic series presented here could be exciting, since the general structure of these equations appear often in gradient and Hessian calculations for a variety of distributions. Finally, application of the DM and BLM distributions could be explored in areas other than the bag-of-words natural language processing classification task, such as problems in bioinformatics and ecology.

4.6 Data availability and extended derivations

4.6.1 Data availability

All code used to generate the data and figures presented here is available online at https://github.com/lakinsm/fast-mle-multinomials and is preserved in its current state under the Publication release.

4.6.2 Extended derivations

Convergence of the beta-Liouville Newton-Raphson procedure

It is sufficient to show that the Hessian matrix, associated with the likelihood function, is negative definite to guarantee a unique global optimum and attain the BLM MLEs. Here we show the form of the Hessian matrix and provide conditions required for it to be negative definite. Continuing from the BLM Hessian described in equations (4.10) and (4.11), the components of the Hessian are,

$$
\begin{aligned}
\frac{\delta^2 F(\boldsymbol{\theta})}{\delta^2 \alpha} &= -\sum_{i=0}^{(\sum_{d=1}^{D} x_d)-1} \frac{1}{(\alpha+i)^2} + \sum_{i=0}^{(\sum_{d=1}^{D+1} x_d)-1} \frac{1}{(\alpha+\beta+i)^2} \\
\frac{\delta^2 F(\boldsymbol{\theta})}{\delta^2 \beta} &= -\sum_{i=0}^{x_{D+1}-1} \frac{1}{(\beta+i)^2} + \sum_{i=0}^{(\sum_{d=1}^{D+1} x_d)-1} \frac{1}{(\alpha+\beta+i)^2} \\
\frac{\delta^2 F(\boldsymbol{\theta})}{\delta \alpha \delta \beta} &= \sum_{i=0}^{(\sum_{d=1}^{D+1} x_d)-1} \frac{1}{(\alpha+\beta+i)^2} \\
\frac{\delta^2 F(\boldsymbol{\theta})}{\delta^2 \alpha_d} &= -\sum_{i=0}^{x_d-1} \frac{1}{(\alpha_d+i)^2} + \sum_{i=0}^{(\sum_{d=1}^{D} x_d)} \frac{1}{(\sum_{d=1}^{D} \alpha_d+i)^2} \\
\frac{\delta^2 F(\boldsymbol{\theta})}{\delta \alpha_d \delta \alpha_g} &= \sum_{i=0}^{(\sum_{d=1}^{D} x_d)} \frac{1}{(\sum_{d=1}^{D} \alpha_d+i)^2} \forall \{d \neq g\}
\end{aligned}
\tag{4.26}
$$

Where,

$$
\begin{aligned}
0 < a_\alpha &= \sum_{i=0}^{(\sum_{d=1}^{D} x_d)-1} \frac{1}{(\alpha+i)^2} \\
0 < a_\beta &= \sum_{i=0}^{x_{D+1}-1} \frac{1}{(\beta+i)^2} \\
0 < c_1 &= \sum_{i=0}^{(\sum_{d=1}^{D+1} x_d)-1} \frac{1}{(\alpha+\beta+i)^2} \\
0 < a_{\alpha_d} &= \sum_{i=0}^{x_d-1} \frac{1}{(\alpha_d+i)^2} \\
0 < c_2 &= \sum_{i=0}^{(\sum_{d=1}^{D} x_d)} \frac{1}{(\sum_{d=1}^{D} \alpha_d+i)^2}
\end{aligned} \tag{4.27}
$$

Then,

$$
\boldsymbol{H}(\boldsymbol{\theta}) = \begin{pmatrix}
-a_\alpha + c_1 & c_1 & 0 & 0 & 0 & \dots \\
c_1 & -a_\beta + c_1 & 0 & 0 & 0 & \dots \\
0 & 0 & -a_{\alpha_1} + c_2 & c_2 & c_2 & \dots \\
0 & 0 & c_2 & -a_{\alpha_2} + c_2 & c_2 & \dots \\
\vdots & & \ddots & & \dots &
\end{pmatrix} \tag{4.28}
$$

This is a symmetric matrix and is negative definite if its pivots are negative. Using Gaussian elimination, the above matrix can be reduced to,

$$
\begin{pmatrix}
-a_\alpha + c_1 & c_1 & 0 & 0 & 0 & 0 & \dots \\
0 & -a_\beta - a_\alpha & 0 & 0 & 0 & 0 & \dots \\
0 & 0 & -a_{\alpha_1} + c_2 & c_2 & c_2 & c_2 & \dots \\
0 & 0 & 0 & -a_{\alpha_2} - a_{\alpha_1} & -a_{\alpha_1} & -a_{\alpha_1} & \dots \\
0 & 0 & 0 & 0 & -a_{\alpha_3} - a_{\alpha_2} & -a_{\alpha_2} & \dots \\
\vdots & & & \ddots & & & \dots
\end{pmatrix} \tag{4.29}
$$

given that $a, c > 0$ for all a and c, all pivots are guaranteed to be negative except for,

$$\begin{aligned} -a_{\alpha} + c_1 \\ -a_{\alpha_1} + c_2 \end{aligned} \tag{4.30}$$

Therefore to obtain a negative definite Hessian, it is sufficient to ascertain that these two terms are negative. That is, the following conditions should be met,

$$\begin{aligned} \sum_{i=0}^{(\sum_{d=1}^{D} x_d)-1} \frac{1}{(\alpha+i)^2} > \sum_{i=0}^{(\sum_{d=1}^{D+1} x_d)-1} \frac{1}{(\alpha+\beta+i)^2} \\ \sum_{i=0}^{x_1-1} \frac{1}{(\alpha_1+i)^2} > \sum_{i=0}^{(\sum_{d=1}^{D} x_d)-1} \frac{1}{(\sum_{d=1}^{D}\alpha_d+i)^2} \end{aligned} \tag{4.31}$$

which are equivalent to,

$$\begin{aligned} \sum_{i=0}^{(\sum_{d=1}^{D} x_d)-1} \left(\frac{1}{(\alpha+i)^2} - \frac{1}{(\alpha+\beta+i)^2} \right) > \sum_{i=\sum_{d=1}^{D} x_d}^{(\sum_{d=1}^{D+1} x_d)-1} \frac{1}{(\alpha+\beta+i)^2} \\ \sum_{i=0}^{x_1-1} \left(\frac{1}{(\alpha_1+i)^2} - \frac{1}{(\sum_{d=1}^{D}\alpha_d+i)^2} \right) > \sum_{i=x_1}^{(\sum_{d=1}^{D} x_d)-1} \frac{1}{(\sum_{d=1}^{D}\alpha_d+i)^2} \end{aligned} \tag{4.32}$$

For the practical application of the BLM in classification, for the purposes of the current paper, we added a small positive number to ensure these conditions are met.

Finite limit approximations

We attempted to approximate the following finite sums of the general form. Though we recognize this is entirely *ad hoc* and does not constitute a formal proof, we provide it for completeness, since it seemed promising in an applied sense and could be used as a starting point for future work.

$$f_1(\theta, N) = \sum_{n=0}^{N} \frac{1}{\theta + n} = \frac{1}{\theta} + \sum_{n=1}^{N} \frac{1}{\theta + n} \tag{4.33}$$

$$f_2(\theta, N) = \sum_{n=0}^{N} \frac{1}{(\theta + n)^2} = \frac{1}{\theta^2} + \sum_{n=1}^{N} \frac{1}{(\theta + n)^2} \tag{4.34}$$

for positive real θ and non-negative integer N. Approximation was performed by starting with the known approximations for the harmonic and geometric series when $\theta \to 0$. The R "nls" function was then used to model various functional forms by trial and error, continually increasing in accuracy by modeling the residuals of each fit. The following are the approximations used in the manuscript for equations 13 and 14, respectively,

$$\begin{aligned} \hat{f}_1(\theta, N) = \frac{1}{\theta} + \frac{\ln(N) + \gamma + \frac{1}{2N} - \frac{1}{12N^2}}{1 + e^{\left(\frac{\ln(\theta) - \frac{\ln(N)}{2} + \frac{e}{10}}{0.072\ln(N)^{1.27} + \frac{0.1677}{1+\ln(N)} + 0.835}\right)}} \\ + A\cos\left(\frac{3\pi}{2} - \left[\frac{4\pi}{1 + e^{\frac{\ln(\theta) - \frac{\ln(N)}{2} + \frac{e}{10}}{v\left(0.072\ln(N)^{1.27} + \frac{0.1677}{1+\ln(N)} + 0.835\right)}}}\right]\right) \end{aligned} \tag{4.35}$$

where γ is the Euler-Mascheroni constant, $A = 0.1068\left(\ln(N)^{0.8224} + \frac{4.986}{\ln(N)+6.408}\right) - 0.7751$, and $v = 3.764e^{\left(\frac{-\pi}{6}\right)(\ln(N)+1)^{1.06}} + 1.59$,

$$\hat{f}_2(\theta, N) = \frac{1}{\theta^2} + \frac{\frac{\pi^2}{6}}{1 + \left(\frac{\pi}{2}e^{\theta}\right)}, \forall N > 200 \tag{4.36}$$

where $\hat{f}_2$ is only accurate for $N > 200$, and all computations that did not meet this criterion were calculated exactly (not using this approximation). Readers familiar with the harmonic and geometric series will recognize that $\ln(N) + \gamma + \frac{1}{2N} - \frac{1}{12N^2}$ is the finite series approximation for the harmonic series, and $\frac{\pi^2}{6}$ is the limit of the geometric series [111, 112]. Examples of the above approximations are implemented in the open source code included with this manuscript (see Data Availability).

Detailed methods for Markov chain Monte-Carlo data simulation

Given a vector of counts $\boldsymbol{x}_c$ for class $c \in C$ and a vector of fixed, generating parameters $\boldsymbol{\theta}_c$,

$$P(\boldsymbol{x}_c|\boldsymbol{\theta}_c) = \frac{P(\boldsymbol{\theta}_c|\boldsymbol{x}_c)P(\boldsymbol{x}_c)}{P(\boldsymbol{\theta}_c)} \tag{4.37}$$

Using a Metropolis-Hastings sampling strategy [117], data were then generated according to the following acceptance probability P_α:

$$\begin{aligned} \alpha &= \min\left\{1, \frac{P(\boldsymbol{x}_{i+1}|\boldsymbol{\theta}_c)q(\boldsymbol{x}_i|\boldsymbol{x}_{i+1})}{P(\boldsymbol{x}_i|\boldsymbol{\theta}_c)q(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i)}\right\} \\ &= \min\left\{1, \frac{P(\boldsymbol{\theta}_c|\boldsymbol{x}_{i+1})P(\boldsymbol{x}_{i+1})q(\boldsymbol{x}_i|\boldsymbol{x}_{i+1})}{P(\boldsymbol{\theta}_c|\boldsymbol{x}_i)P(\boldsymbol{x}_i)q(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i)}\right\} \end{aligned} \tag{4.38}$$

Taking $q(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i) = P(\boldsymbol{x}_{i+1})$,

$$\begin{aligned} \alpha &= \min\left\{1, \frac{P(\boldsymbol{\theta}_c|\boldsymbol{x}_{i+1})}{P(\boldsymbol{\theta}_c|\boldsymbol{x}_i)}\right\} \\ &= \min\left\{1, \frac{P(\boldsymbol{x}_{i+1}|\boldsymbol{\theta}_c)/P(\boldsymbol{x}_{i+1})}{P(\boldsymbol{x}_i|\boldsymbol{\theta}_c)/P(\boldsymbol{x}_i)}\right\} \end{aligned} \tag{4.39}$$

We used a multinomial distribution for the proposal $P(\boldsymbol{x})$. We then sampled from a distribution $U \sim \text{uniform}(0, 1)$ and accept any new sample $\boldsymbol{x}_{i+1}$ if $U \leq \alpha$ and we keep $\boldsymbol{x}_i$ otherwise. MCMC simulation was performed separately for each class using a Metropolis-Hastings sampling strategy with a multinomial proposal, $P(\boldsymbol{x})$, and acceptance-rejection probability,

$$\alpha = \min\left\{1, \frac{P(\boldsymbol{x}_{i+1}|\boldsymbol{\theta}_c)/P(\boldsymbol{x}_{i+1})}{P(\boldsymbol{x}_i|\boldsymbol{\theta}_c)/P(\boldsymbol{x}_i)}\right\} \tag{4.40}$$

where $P(\boldsymbol{x}_{i+1}|\boldsymbol{\theta}_c)$ is the likelihood of $\boldsymbol{x}_{i+1}$ under the BLM and $P(\boldsymbol{x}_{i+1})$ is the likelihood of the data under the proposal.

Proof for the use of BLM MLEs for classification on the multinomial simplex

Let there be a $D+1$ dimensional vector with a single count at d, $\boldsymbol{y} = \{0, 0, 0, \ldots, 1, \ldots, 0\}$. We aim to assign this vector to a class $c \in C$. Assume that we have the data matrix $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_C\}$. Such that $\boldsymbol{x}_c = \{x_{c1}, \ldots, x_{cd}, \ldots, x_{C(D+1)}\}$. Let $\boldsymbol{\theta}$ be the $D+2$ parameters of a BLM distribution and $\boldsymbol{p}$ be the $D+1$ parameters of a multinomial distribution. Initially, we assume $\boldsymbol{\theta}$ are known. Additionally, we assume a $0/1$ loss function that will assign $\boldsymbol{y}$ to class c if $P(c|\boldsymbol{y}, \boldsymbol{X})$ is maximum. Accordingly, we want to find,

$$P(c|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}) = \frac{P(\boldsymbol{y}|c, \boldsymbol{X}, \boldsymbol{\theta})P(c|\boldsymbol{X}, \boldsymbol{\theta})}{\sum_C P(\boldsymbol{y}|c, \boldsymbol{X}, \boldsymbol{\theta})P(c|\boldsymbol{X}, \boldsymbol{\theta})} \tag{4.41}$$

Assuming that the classes have equal priors that do not depend on the data and that the classes are independent,

$$P(c|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}) = \frac{P(\boldsymbol{y}|\boldsymbol{x}_c, \boldsymbol{\theta}_c)}{\sum_C P(\boldsymbol{y}|\boldsymbol{x}_c, \boldsymbol{\theta}_c)} \tag{4.42}$$

Given that the denominator of (4.42) is constant,

$$P(c|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}) \propto P(\boldsymbol{y}|\boldsymbol{x}_c, \boldsymbol{\theta}_c) \tag{4.43}$$

Assuming that we have produced BLM MLEs, $\hat{\boldsymbol{\theta}}_c$, utilizing the data $\boldsymbol{x}_c$, and that they are sufficient statistics then,

$$P(c|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}) \propto P(\boldsymbol{y}|\hat{\boldsymbol{\theta}}_c) \tag{4.44}$$

Using (4.4) we find,

$$P(\boldsymbol{y}|\hat{\boldsymbol{\theta}}_c) = \frac{\Gamma\left(\sum_{d=1}^{D}\hat{\alpha}_{cd}\right)\Gamma(\hat{\alpha}_c+\hat{\beta}_c)}{\Gamma(\hat{\alpha}_c)\Gamma(\hat{\beta}_c)\prod_{d=1}^{D}\Gamma(\hat{\alpha}_{cd})}\frac{\Gamma\left(\left(\sum_{d=1}^{D+1}y_d\right)+1\right)}{\prod_{d=1}^{D+1}\Gamma(y_d+1)} \times\frac{\Gamma(\hat{\alpha}_c+\sum_{d=1}^{D}y_d)\Gamma(\hat{\beta}_c+y_{D+1})}{\Gamma(\hat{\alpha}_c+\hat{\beta}_c+\sum_{d=1}^{D+1}y_d)}\frac{\prod_{d=1}^{D}\Gamma(\hat{\alpha}_{cd}+y_d)}{\Gamma(\sum_{d=1}^{D}\hat{\alpha}_{cd}+y_d)} \tag{4.45}$$

For the $\boldsymbol{y} = \{0, 0, 0, \ldots, 1, \ldots, 0, 0, 0\}$ with $y_d = 1$ and by noting that $\Gamma(n+1) = n\Gamma(n)$ we get, for $d \neq D+1$,

$$P(y_d|\hat{\boldsymbol{\theta}}_c) = \hat{p}_d = \frac{\hat{\alpha}_c}{\hat{\alpha}_c+\hat{\beta}_c}\frac{\hat{\alpha}_{cd}}{\sum_{d=1}^{D}\hat{\alpha}_{cd}} \tag{4.46}$$

and for $d = D+1$,

$$P(y_d|\hat{\boldsymbol{\theta}}_c) = \hat{p}_{D+1} = \frac{\hat{\beta}_c}{\hat{\alpha}_c+\hat{\beta}_c} \tag{4.47}$$

We can now use these as the parameters $\boldsymbol{p}$ in a multinomial, assuming independence of the vectors $\boldsymbol{y}$, which results in,

$$P(\boldsymbol{y}|\hat{\boldsymbol{\theta}}_c) = \left(\frac{\hat{\beta}_c}{\hat{\alpha}_c+\hat{\beta}_c}\right)^{y_{D+1}}\prod_{d=1}^{D}\left(\frac{\hat{\alpha}_c}{\hat{\alpha}_c+\hat{\beta}_c}\frac{\hat{\alpha}_{cd}}{\sum_{d=1}^{D}\hat{\alpha}_{cd}}\right)^{y_d} \tag{4.48}$$

where $y_d \geq 0$.

Gold Standard Data Preprocessing

All datasets were preprocessed by Cardoso-Cachopo 2007 as follows:

- Tab, newline, and return characters were replaced with a space.
- Only letters were kept; special characters and numbers were removed.
- All letters were modified to lowercase.

Additionally, all English datasets (excluding Cade12) were preprocessed as follows:

- Words less than 3 characters in length were removed.
- All 524 English stopwords from the SMART Retrieval System were removed [118].
- The remaining words were processed using the Porter Stemming Algorithm [119].

The post-processed datasets are available in the open source GitHub repository associated with this publication (see Data Availability).

4.6.3 Additional figures

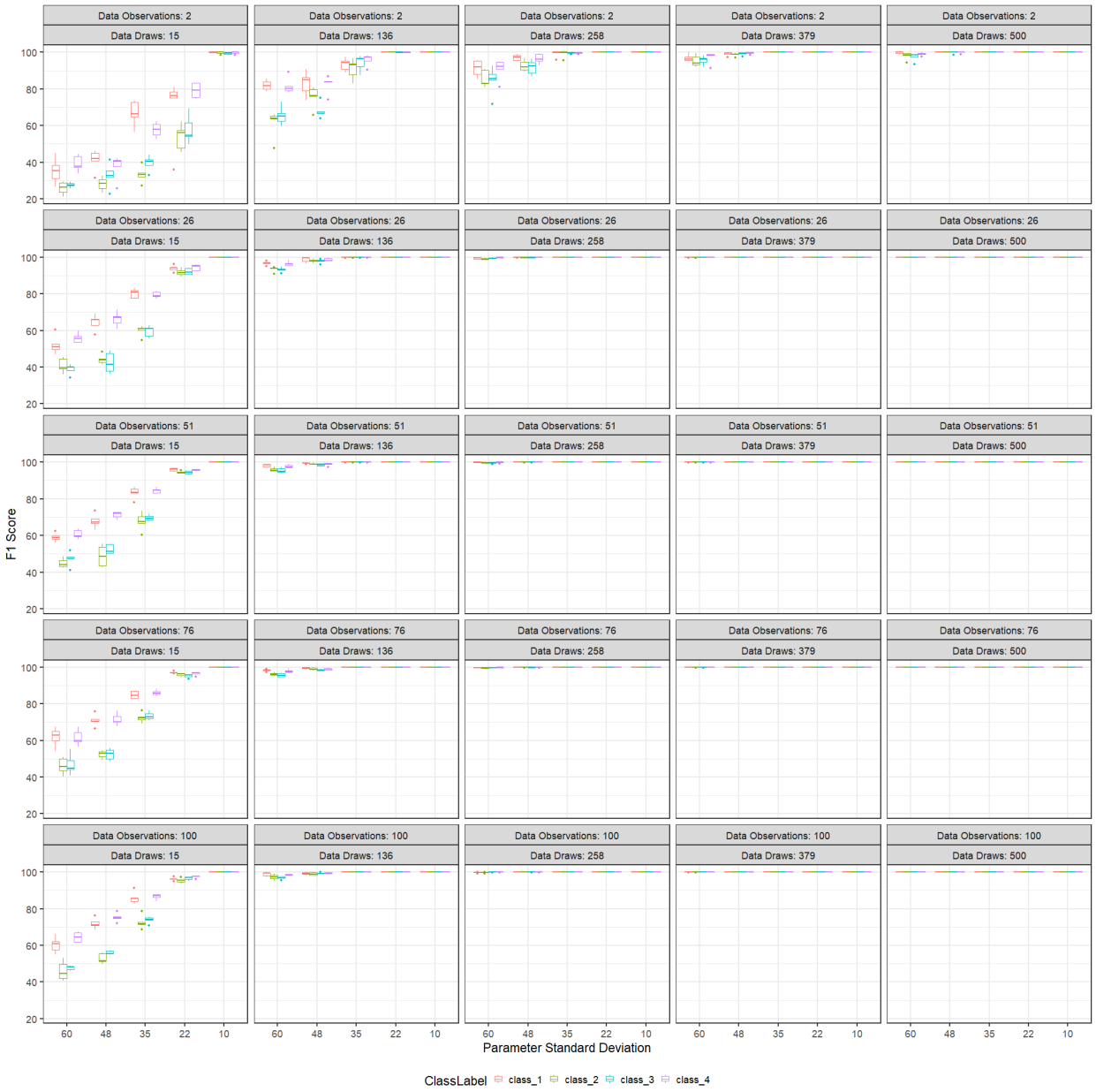

Figure 4.4: Use of the DM distribution results in high F1 classification scores as data draws and observations increase, even for difficult classification tasks. A separate boxplot is shown for each simulated class distribution (classes 1-4). The boxplots summarize the F1 score quartiles for 5 independent datasets for each combination of variables.

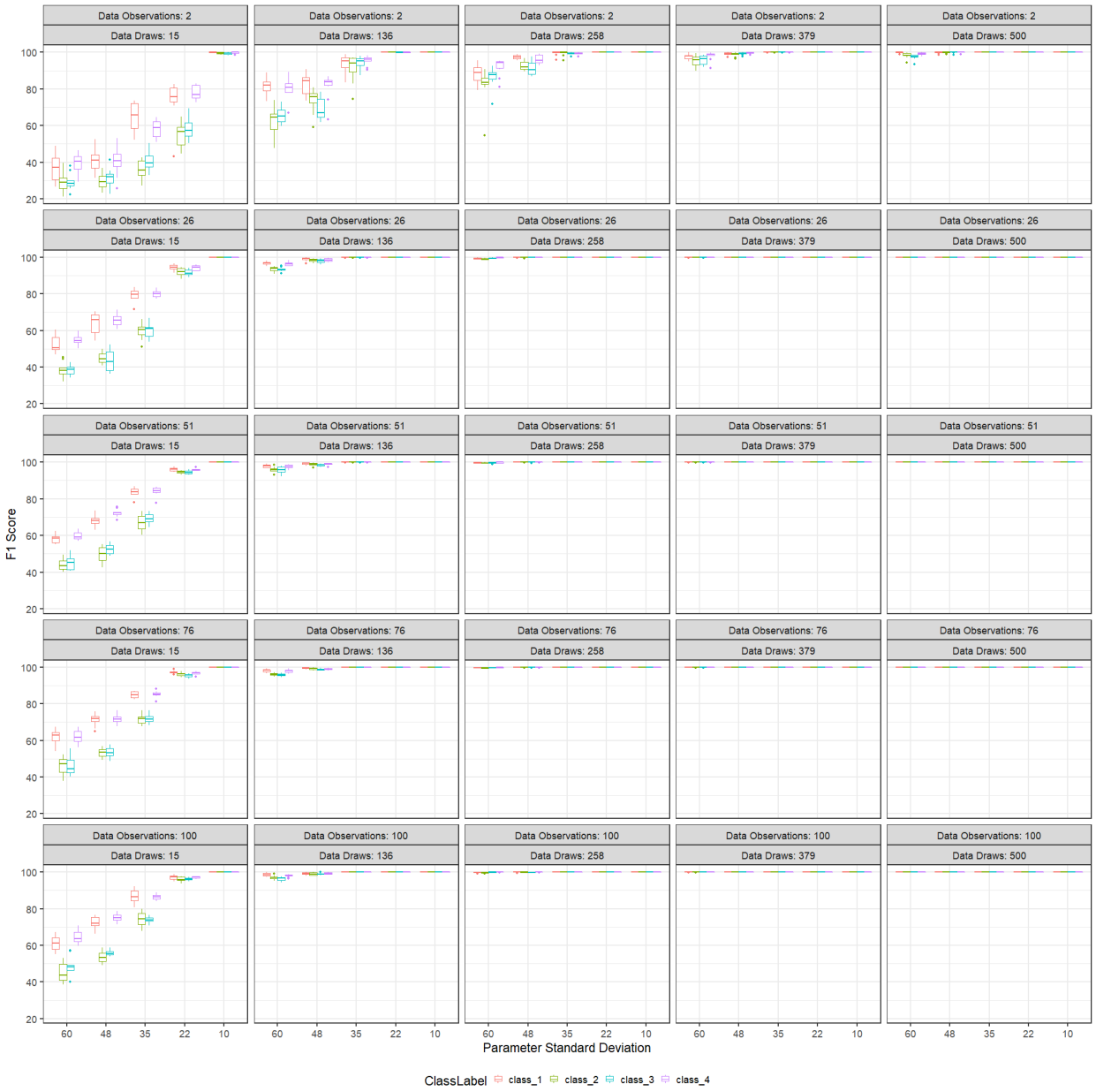

Figure 4.5: Use of the multinomial distribution and standard multinomial naive Bayes results in high F1 classification scores as data draws and observations increase, even for difficult classification tasks. A separate boxplot is shown for each simulated class distribution (classes 1-4). The boxplots summarize the F1 score quartiles for 5 independent datasets for each combination of variables.

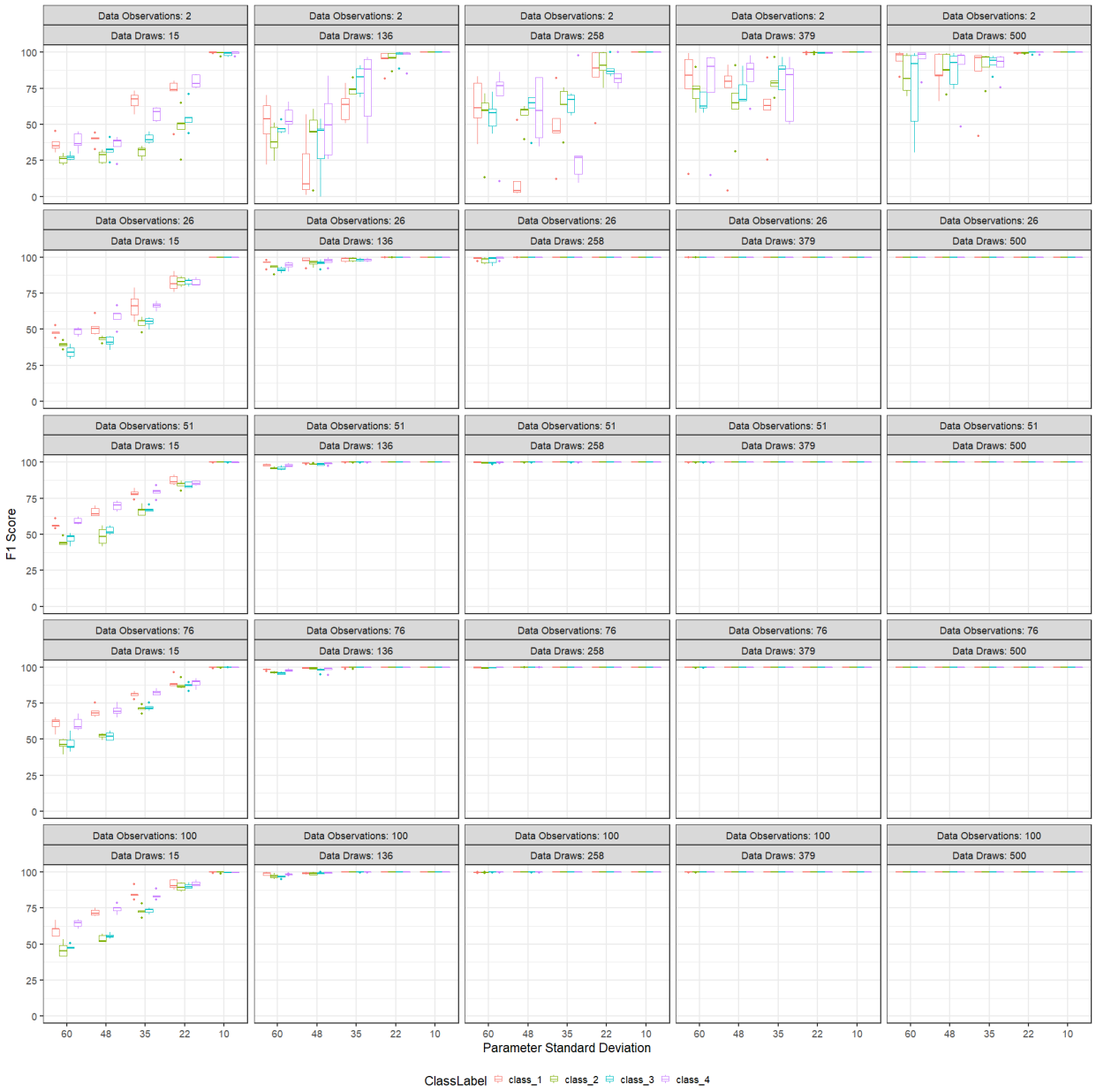

Figure 4.6: Use of the DM distribution with marginal likelihood classification results in high F1 classification scores as data draws and observations increase, even for difficult classification tasks. However, with the marginal likelihood classification methods, more data is required to produce classification results with similar F1 scores to the naive Bayes approaches. A separate boxplot is shown for each simulated class distribution (classes 1-4). The boxplots summarize the F1 score quartiles for 5 independent datasets for each combination of variables.

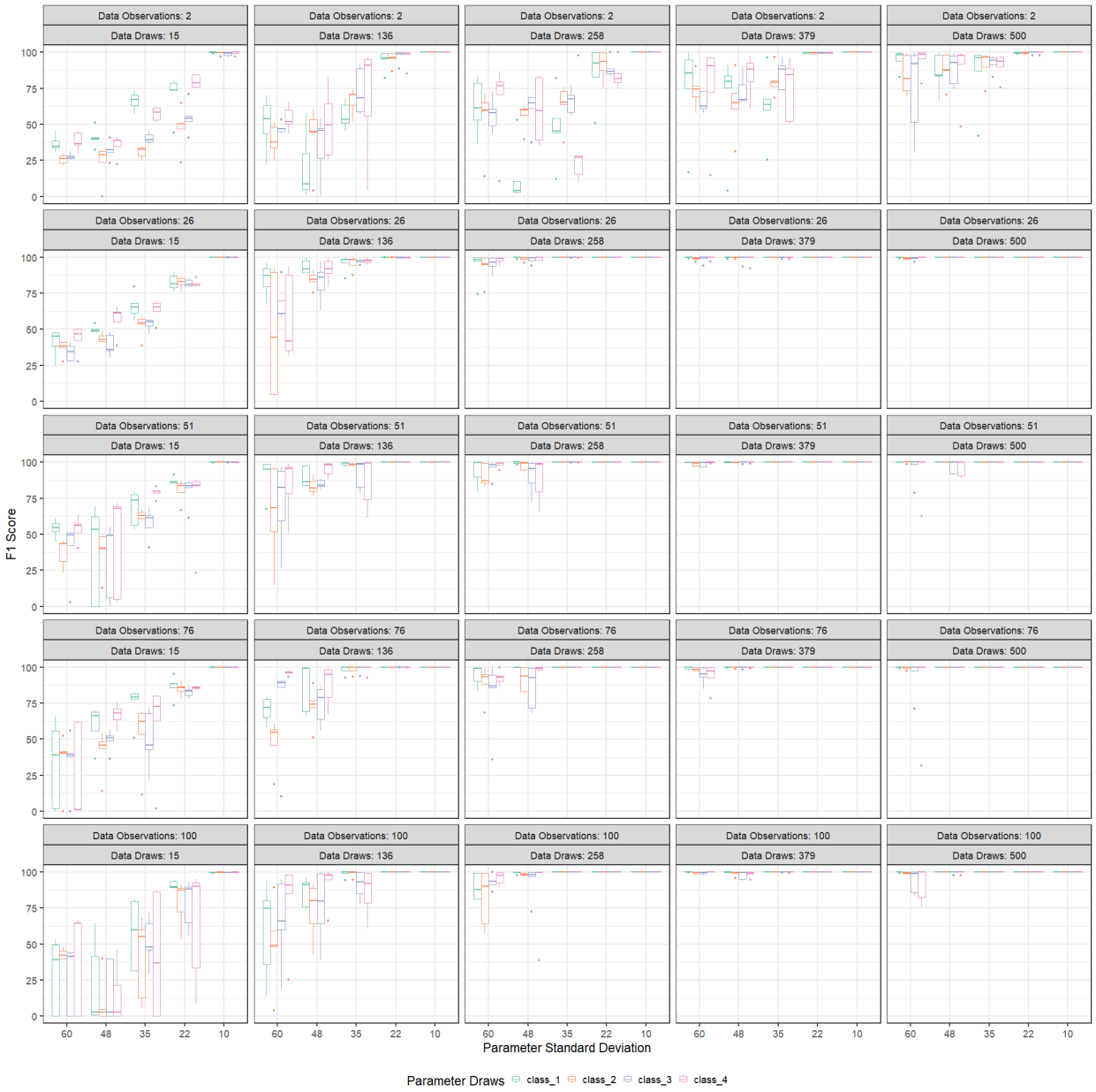

Figure 4.7: Use of the BLM distribution with marginal likelihood classification results in high F1 classification scores as data draws and observations increase, even for difficult classification tasks. However, with the marginal likelihood classification methods, more data is required to produce classification results with similar F1 scores to the naive Bayes approaches. A separate boxplot is shown for each simulated class distribution (classes 1-4). The boxplots summarize the F1 score quartiles for 5 independent datasets for each combination of variables.

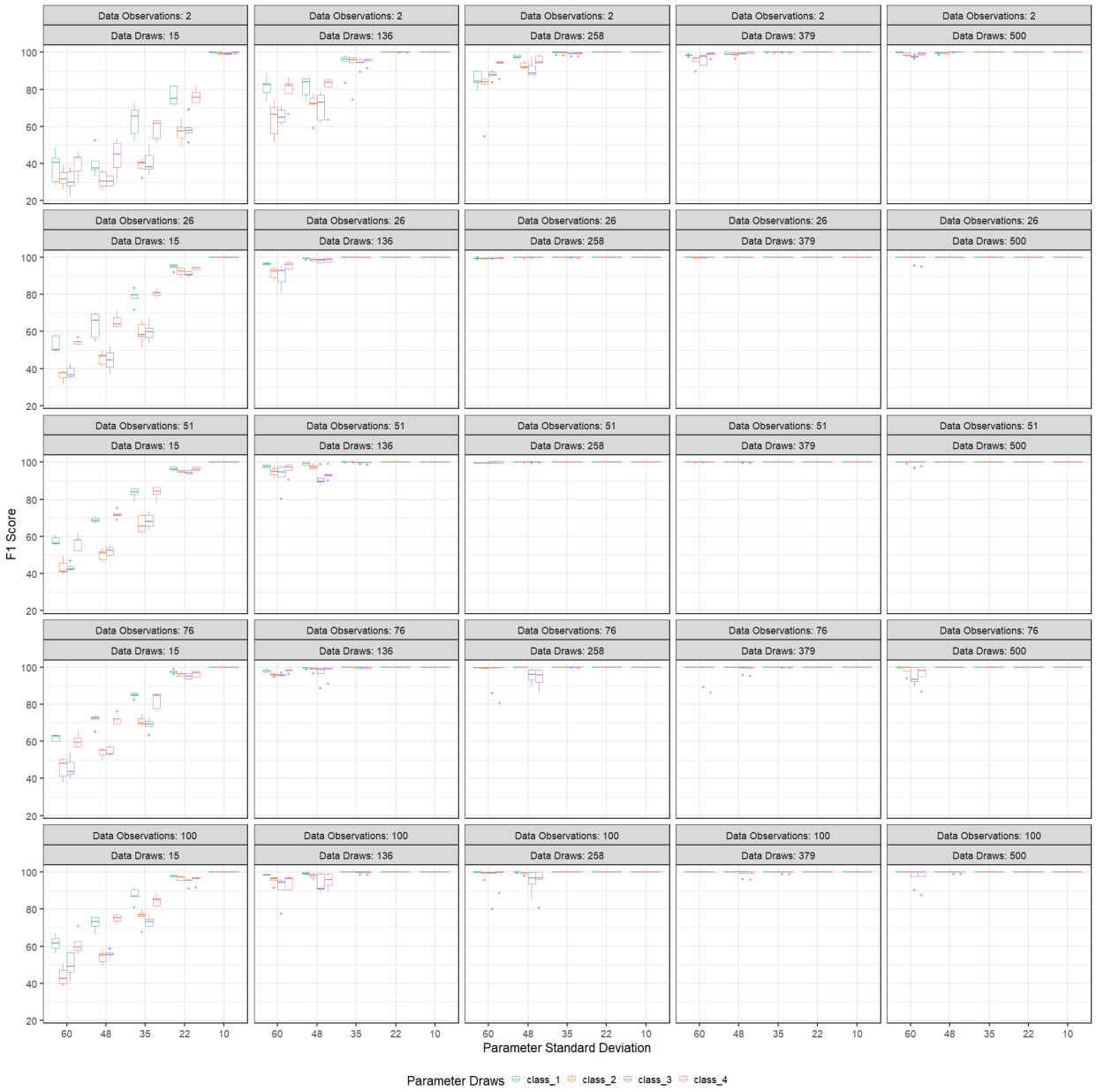

Figure 4.8: On data simulated from the BLM distribution, use of the BLM distribution results in high F1 classification scores as data draws and observations increase, even for difficult classification tasks. There doesn't appear to be a difference between classification scores on multinomial data compared to BLM-simulated data. A separate boxplot is shown for each simulated class distribution (classes 1-4). The boxplots summarize the F1 score quartiles for 5 independent datasets for each combination of variables.

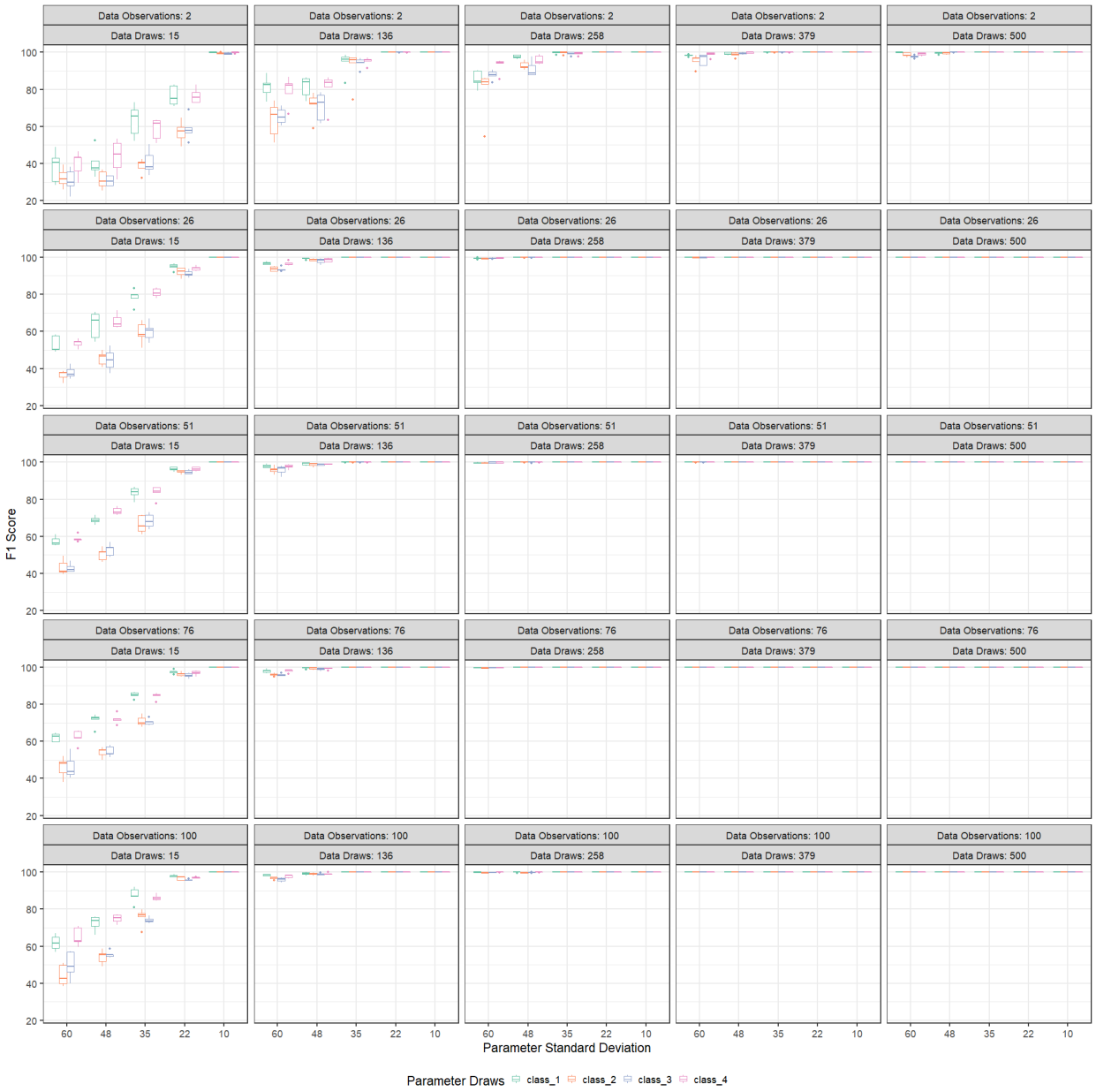

Figure 4.9: On data simulated from the BLM distribution, use of the DM distribution results in high F1 classification scores as data draws and observations increase, even for difficult classification tasks. There doesn't appear to be a difference between classification scores on multinomial data compared to BLM-simulated data. A separate boxplot is shown for each simulated class distribution (classes 1-4). The boxplots summarize the F1 score quartiles for 5 independent datasets for each combination of variables.

Chapter 5

Conclusion

This work has successfully adapted the naive Bayes classification strategy to perform modern genetic sequence classification in the general case. The novel contributions of this work include: a comparison of generalized naive Bayes and outgroup sequence classification against cutting-edge classification tools (Chapter 2); an adaptation of the naive Bayes algorithm into a heavily parallelized framework with schema for extension into distributed computing architectures (Chapter 3); and an exploration of the underlying statistical distributions used in the Bayesian likelihood formulation of the naive Bayes classifier (Chapter 4). Overall, this work has shown that the naive Bayes algorithm is capable of performing as well as modern classifiers for genetic sequence classification in the average case, with marginally better or worse performance in various nuanced classification tasks.

Perhaps the most interesting result of this work is that this implementation of naive Bayes performed similarly to modern classifiers. The similar performance of the methods examined here suggests that the structure of the underlying classification problem contributes more to the outcome than does the choice of classifier itself. For genetic sequence classifiers, this suggests that the methods used to encode information from raw genetic sequences into a feature space fail to capture enough information for machine learning to improve over hash table algorithms and local alignment. If true, this would mean that gains in classification performance via improvement of algorithms alone would be marginal at best, since all available information in the current feature space is already being used to nearly its fullest extent. Therefore, research into feature encoding and feature spaces (or algorithms that transform the feature space) may prove more valuable in increasing the performance of genetic sequence classifiers than focusing efforts exclusively on algorithms, at this point in time.

Much of the current work in genetic sequence classification utilizes some variant of k-mer feature spaces [47, 120, 121]. k-mers have the advantage of being simple to encode, easy to

conceptualize, and computationally efficient to use. However, as shown in Chapter 2, they result in feature spaces that are sparse (with many zero counts) and flat (with few counts greater than 1). Depending on the biological system under study, this may result in the unwanted encoding of many features that are uninformative to the classification task. Classifiers are inundated with noise in such a scenario, since the exponential expansion of the feature space greatly reduces the signal to noise ratio as k increases.

Recent work has therefore focused on feature selection within *k*-mer encoding to improve the signal to noise ratio, including the use of genomic masking of repetitive regions and the employment of so-called minimizers that capture *k*-mer information in a more compressed space [121, 122]. While these have the advantage of improving the signal to noise ratio, they do not improve the total amount of signal present in the feature space (i.e. they remove noise but do not increase signal). Thus, it may be worthwhile to explore enhancements to the *k*-mer feature space or an alternative feature encoding strategy that increases the signal strength for classification. Enhancements to *k*-mers could include making Markovian assumptions over *k*-mers, such that *k*-mers are no longer considered to be independent of one another. This would incorporate elements of hidden Markov models or recurrent neural networks that have proven effective in other fields like natural language processing (NLP) [73, 123]. Alternatively, information from phylogenetics, such as the PAM and BLOSUM matrices, could be layered over *k*-mer encoding such that we only use *k*-mers that are informative to the evolutionary system under study [1]. In the field of NLP that deals with human language, gibberish words are not encoded or considered during classification tasks; so why should random or gibberish *k*-mers be used for genetic sequence classification?

This harkens back to the discussion of genetic tensors outlined in the Introduction. Clearly, not all paths between two points in a genetic topology are weighted equally, since selective pressures have a large impact on what genetic sequences are viable from an evolutionary standpoint. Thus, it may be the case that to improve upon genetic sequence classification going forward, we must find a way to incorporate genetic distance information into the feature space, such that classifiers can utilize it appropriately. In closing, the work I have presented here seems to support this line

of investigation into genetic feature spaces, and I hope that it will serve as a foundation for that purpose.

References

[1] M. Dayhoff and R. M. Schwartz, "Matrices for detecting distant relationships," *Atlas of Protein Sequence and Structure*, vol. 5, pp. 353–358, 1978.

[2] S. Y. W. Ho and S. Duchêne, "Molecular-clock methods for estimating evolutionary rates and timescales," *Molecular Ecology*, vol. 23, no. 24, pp. 5947–5965, 2014. DOI: 10.1111/mec.12953.

[3] O. Mayo, "A century of Hardy–Weinberg equilibrium," *Twin Research and Human Genetics*, vol. 11, no. 3, pp. 249–256, 2008. DOI: 10.1375/twin.11.3.249.

[4] 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," *Applied and Environmental Microbiology*, vol. 73, no. 16, pp. 5261–5267, 2007.

[5] M. J. Alvarez-Cubero, M. Saiz, B. Martínez-García, S. M. Sayalero, C. Entrala, *et al.*, "Next generation sequencing: An application in forensic sciences?" *Annals of Human Biology*, vol. 44, no. 7, pp. 581–592, 2017. DOI: 10.1080/03014460.2017.1375155.

[6] S. Yohe and B. Thyagarajan, "Review of clinical next-generation sequencing," *Archives of Pathology and Laboratory Medicine*, vol. 141, no. 11, pp. 1544–1557, 2017. DOI: 10.5858/arpa.2016-0501-RA.

[7] W. Gu, S. Miller, and C. Y. Chiu, "Clinical metagenomic next-generation sequencing for pathogen detection," *Annual Review of Pathology: Mechanisms of Disease*, vol. 14, no. 1, pp. 319–338, 2019. DOI: 10.1146/annurev-pathmechdis-012418-012751.

[8] B. Jagadeesan, P. Gerner-Smidt, M. W. Allard, S. Leuillet, A. Winkler, *et al.*, "The use of next generation sequencing for improving food safety: Translation into practice," *Food Microbiology*, vol. 79, pp. 96–115, 2019, ISSN: 0740-0020. DOI: 10.1016/j.fm.2018.11.005.

[9] B. Korber, W. M. Fischer, S. Gnanakaran, H. Yoon, J. Theiler, *et al.*, "Tracking changes in SARS-CoV-2 Spike: Evidence that D614G increases infectivity of the COVID-19 virus," *Cell*, vol. 182, no. 4, 812–827.e19, Aug. 2020, ISSN: 0092-8674. DOI: 10.1016/j.cell.2020.06.043.

[10] R. P. McNamara, C. Caro-Vegas, J. T. Landis, R. Moorad, L. J. Pluta, *et al.*, "High-density amplicon sequencing identifies community spread and ongoing evolution of SARS-CoV-2 in the Southern United States," *Cell Reports*, vol. 33, no. 5, Nov. 2020, ISSN: 2211-1247. DOI: 10.1016/j.celrep.2020.108352.

[11] M. Boolchandani, A. W. D'Souza, and G. Dantas, "Sequencing-based methods and resources to study antimicrobial resistance," *Nature Reviews Genetics*, vol. 20, no. 6, pp. 356–370, 2019.

[12] M. Otto, "Next-generation sequencing to monitor the spread of antimicrobial resistance," *Genome Medicine*, vol. 9, no. 1, p. 68, Jul. 2017. DOI: 10.1186/s13073-017-0461-x.

[13] D. M. Hyman, D. B. Solit, M. E. Arcila, D. T. Cheng, P. Sabbatini, *et al.*, "Precision medicine at Memorial Sloan Kettering Cancer Center: clinical next-generation sequencing enabling next-generation targeted therapy trials," *Drug Discovery Today*, vol. 20, no. 12, pp. 1422–1428, 2015, SI: Precision medicine, ISSN: 1359-6446. DOI: 10.1016/j.drudis.2015.08.005.

[14] C. N. Kline, N. M. Joseph, J. P. Grenert, J. van Ziffle, E. Talevich, *et al.*, "Targeted next-generation sequencing of pediatric neuro-oncology patients improves diagnosis, identifies pathogenic germline mutations, and directs targeted therapy," *Neuro-Oncology*, vol. 19, no. 5, pp. 699–709, Nov. 2016. DOI: 10.1093/neuonc/now254.

[15] H. Li and R. Durbin, "Fast and accurate long-read alignment with Burrows-Wheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.

[16] D. E. Wood, J. Lu, and B. Langmead, "Improved metagenomic analysis with Kraken 2," *Genome biology*, vol. 20, no. 1, pp. 1–13, 2019.

[17] E. Doster, S. M. Lakin, C. J. Dean, C. Wolfe, J. G. Young, *et al.*, "MEGARes 2.0: a database for classification of antimicrobial drug, biocide and metal resistance determinants in metagenomic sequence data," *Nucleic Acids Research*, vol. 48, no. D1, pp. D561–D569, 2020.

[18] T. Laver, J. Harrison, P. O'Neill, K. Moore, A. Farbos, *et al.*, "Assessing the performance of the Oxford Nanopore Technologies MinION," *Biomolecular Detection and Quantification*, vol. 3, pp. 1–8, 2015. DOI: 10.1016/j.bdq.2015.02.001.

[19] D. R. Greig, C. Jenkins, S. Gharbia, and T. J. Dallman, "Comparison of single-nucleotide variants identified by Illumina and Oxford Nanopore technologies in the context of a potential outbreak of Shiga toxin–producing Escherichia coli," *GigaScience*, vol. 8, no. 8, Aug. 2019. DOI: 10.1093/gigascience/giz104.

[20] M. O. Pollard, D. Gurdasani, A. J. Mentzer, T. Porter, and M. S. Sandhu, "Long reads: their purpose and place," *Human Molecular Genetics*, vol. 27, no. R2, R234–R241, May 2018. DOI: 10.1093/hmg/ddy177.

[21] K. Deiner, M. A. Renshaw, Y. Li, B. P. Olds, D. M. Lodge, *et al.*, "Long-range PCR allows sequencing of mitochondrial genomes from environmental DNA," *Methods in Ecology and Evolution*, vol. 8, no. 12, pp. 1888–1898, 2017. DOI: 10.1111/2041-210X.12836.

[22] Y. Matsuo, S. Komiya, Y. Yasumizu, Y. Yasuoka, K. Mizushima, *et al.*, "Full-length 16S rRNA gene amplicon analysis of human gut microbiota using MinION™ nanopore sequencing confers species-level resolution," *BMC Microbiology*, vol. 21, no. 1, p. 35, Jan. 2021. DOI: 10.1186/s12866-021-02094-5.

[23] J. Hess, T. Kohl, M. Kotrová, K. Rönsch, T. Paprotka, *et al.*, "Library preparation for next generation sequencing: A review of automation strategies," *Biotechnology Advances*, vol. 41, p. 107 537, 2020. DOI: 10.1016/j.biotechadv.2020.107537.

[24] E. R. Mardis, "DNA sequencing technologies: 2006-2016," *Nature Protocols*, vol. 12, no. 2, p. 213, 2017.

[25] E. L. van Dijk, Y. Jaszczyszyn, D. Naquin, and C. Thermes, "The third revolution in sequencing technology," *Trends in Genetics*, vol. 34, no. 9, pp. 666–681, 2018, ISSN: 0168-9525. DOI: 10.1016/j.tig.2018.05.008.

[26] G. Tini, L. Marchetti, C. Priami, and M.-P. Scott-Boyer, "Multi-omics integration—a comparison of unsupervised clustering methodologies," *Briefings in Bioinformatics*, vol. 20, no. 4, pp. 1269–1279, Dec. 2017. DOI: 10.1093/bib/bbx167.

[27] B. J. Callahan, P. J. McMurdie, M. J. Rosen, A. W. Han, A. J. A. Johnson, *et al.*, "DADA2: high-resolution sample inference from Illumina amplicon data," *Nature Methods*, vol. 13, no. 7, pp. 581–583, 2016.

[28] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li, "CD-HIT: accelerated for clustering the next-generation sequencing data," *Bioinformatics*, vol. 28, no. 23, pp. 3150–3152, Dec. 2012. DOI: 10.1093/bioinformatics/bts565.

[29] Y.-H. Kim, Y. Song, J.-K. Kim, T.-M. Kim, H. W. Sim, *et al.*, "False-negative errors in next-generation sequencing contribute substantially to inconsistency of mutation databases," *PLOS ONE*, vol. 14, no. 9, pp. 1–14, Sep. 2019. DOI: 10.1371/journal.pone.0222535.

[30] F. Pompanon and S. Samadi, "Next generation sequencing for characterizing biodiversity: Promises and challenges," *Genetica*, vol. 143, no. 2, pp. 133–138, Apr. 2015. DOI: 10.1007/s10709-015-9816-7.

[31] R. D. Stewart, M. D. Auffret, A. Warr, A. W. Walker, R. Roehe, *et al.*, "Compendium of 4,941 rumen metagenome-assembled genomes for rumen microbiome biology and enzyme discovery," *Nature Biotechnology*, vol. 37, no. 8, pp. 953–961, Aug. 2019. DOI: 10.1038/s41587-019-0202-3.

[32] K. C. Nixon and Q. D. Wheeler, "An amplification of the phylogenetic species concept," *Cladistics*, vol. 6, no. 3, pp. 211–223, 1990. DOI: 10.1111/j.1096-0031.1990.tb00541.x.

[33] R. D. Dowell, O. Ryan, A. Jansen, D. Cheung, S. Agarwala, *et al.*, "Genotype to phenotype: A complex problem," *Science*, vol. 328, no. 5977, pp. 469–469, 2010. DOI: 10.1126/science.1189015.

[34] D. Kim, L. Song, F. P. Breitwieser, and S. L. Salzberg, "Centrifuge: Rapid and sensitive classification of metagenomic sequences," *Genome research*, vol. 26, no. 12, pp. 1721–1729, 2016.

[35] N. Segata, L. Waldron, A. Ballarini, V. Narasimhan, O. Jousson, *et al.*, "Metagenomic microbial community profiling using unique clade-specific marker genes," *Nature Methods*, vol. 9, no. 8, pp. 811–814, 2012.

[36] M. S. Rahman *et al.*, *Basic graph theory*. Springer, 2017.

[37] J. A. Blake and C. J. Bult, "Beyond the data deluge: Data integration and bio-ontologies," *Journal of Biomedical Informatics*, vol. 39, no. 3, pp. 314–320, 2006, Biomedical Ontologies. DOI: https://doi.org/10.1016/j.jbi.2006.01.003.

[38] F. Gutierrez, "Semantic technologies and bio-ontologies," in *Bioinformatics in MicroRNA Research*, J. Huang, G. M. Borchert, D. Dou, J. (Huan, W. Lan, *et al.*, Eds. New York, NY: Springer New York, 2017, pp. 83–91. DOI: 10.1007/978-1-4939-7046-9_6.

[39] P. J. McMurdie and S. Holmes, "Waste not, want not: Why rarefying microbiome data is inadmissible," *PLOS Computational Biology*, vol. 10, no. 4, pp. 1–12, Apr. 2014. DOI: 10.1371/journal.pcbi.1003531.

[40] S. Weiss, Z. Z. Xu, S. Peddada, A. Amir, K. Bittinger, *et al.*, "Normalization and microbial differential abundance strategies depend upon data characteristics," *Microbiome*, vol. 5, no. 1, p. 27, Mar. 2017. DOI: 10.1186/s40168-017-0237-y.

[41] V. J. Promponas, I. Iliopoulos, and C. A. Ouzounis, "Annotation inconsistencies beyond sequence similarity-based function prediction - phylogeny and genome structure," *Standards in Genomic Sciences*, vol. 10, pp. 108–108, Nov. 2015. DOI: 10.1186/s40793-015-0101-2.

[42] E. Doster, P. Rovira, N. R. Noyes, B. A. Burgess, X. Yang, *et al.*, "A cautionary report for pathogen identification using shotgun metagenomics; a comparison to aerobic culture and polymerase chain reaction for salmonella enterica identification," *Frontiers in Microbiology*, vol. 10, p. 2499, 2019. DOI: 10.3389/fmicb.2019.02499.

[43] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17, Chicago, Illinois, USA: Association for Computing Machinery, 2017, pp. 1009–1024, ISBN: 9781450341974. DOI: 10.1145/3035918.3064029.

[44] D. Jankov, S. Luo, B. Yuan, Z. Cai, J. Zou, *et al.*, "Declarative recursive computation on an rdbms," *Proceedings of the VLDB Endowment*, vol. 12, no. 7, pp. 822–835, Mar. 2019, ISSN: 2150-8097. DOI: 10.14778/3317315.3317323.

[45] A. Malte and P. Ratadiya, *Evolution of transfer learning in natural language processing*, 2019. arXiv: 1910.07370 [cs.CL].

[46] I.-M. A. Chen, K. Chu, K. Palaniappan, M. Pillay, A. Ratner, *et al.*, "IMG/M v.5.0: an integrated data management and comparative analysis system for microbial genomes and microbiomes," *Nucleic Acids Research*, vol. 47, no. D1, pp. D666–D677, Oct. 2018. DOI: 10.1093/nar/gky901.

[47] C. Marchet, C. Boucher, S. J. Puglisi, P. Medvedev, M. Salson, *et al.*, "Data structures based on k-mers for querying large collections of sequencing data sets," *Genome Research*, 2020. DOI: 10.1101/gr.260604.119.

[48] E. Asgari and M. R. Mofrad, "Continuous distributed representation of biological sequences for deep proteomics and genomics," *PLOS ONE*, vol. 10, no. 11, e0141287, 2015.

[49] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE.," *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.

[50] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[51] W. J. Kent, "BLAT—The BLAST-Like Alignment Tool," *Genome Research*, vol. 12, no. 4, pp. 656–664, 2002. DOI: 10.1101/gr.229202.

[52] L. Wang and T. Jiang, "On the complexity of multiple sequence alignment," *Journal of Computational Biology*, vol. 1, no. 4, pp. 337–348, 1994. DOI: 10.1089/cmb.1994.1.337.

[53] M. Holtgrewe, A.-K. Emde, D. Weese, and K. Reinert, "A novel and well-defined benchmarking method for second generation read mapping," *BMC Bioinformatics*, vol. 12, no. 1, pp. 1–10, 2011.

[54] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2000, p. 390. DOI: 10.1109/SFCS.2000.892127.

[55] P. Ferragina and G. Manzini, "Indexing compressed text," *Journal of the ACM*, vol. 52, no. 4, pp. 552–581, Jul. 2005. DOI: 10.1145/1082036.1082039.

[56] D. E. Wood and S. L. Salzberg, "Kraken: Ultrafast metagenomic sequence classification using exact alignments," *Genome Biology*, vol. 15, no. 3, pp. 1–12, 2014.

[57] K. Brinda, M. Sykulski, and G. Kucherov, "Spaced seeds improve k-mer-based metagenomic classification," *Bioinformatics*, vol. 31, no. 22, pp. 3584–3592, Jul. 2015, ISSN: 1460-2059. DOI: 10.1093/bioinformatics/btv419.

[58] N. G. Nguyen, V. A. Tran, D. L. Ngo, D. Phan, F. R. Lumbanraja, *et al.*, "DNA sequence classification by convolutional neural network," *Journal of Biomedical Science and Engineering*, vol. 9, no. 05, p. 280, 2016.

[59] Y. Liu, J. Guo, G. Hu, and H. Zhu, "Gene prediction in metagenomic fragments based on the SVM algorithm," in *BMC Bioinformatics*, Springer, vol. 14, 2013, pp. 1–12.

[60] S. Salzberg, A. L. Delcher, K. H. Fasman, and J. Henderson, "A decision tree system for finding genes in DNA," *Journal of Computational Biology*, vol. 5, no. 4, pp. 667–680, 1998.

[61] N. A. Bokulich, B. D. Kaehler, J. R. Rideout, M. Dillon, E. Bolyen, *et al.*, "Optimizing taxonomic classification of marker-gene amplicon sequences with QIIME 2's q2-feature-classifier plugin," *Microbiome*, vol. 6, no. 1, pp. 1–17, 2018.

[62] M. Escalona, S. Rocha, and D. Posada, "A comparison of tools for the simulation of genomic next-generation sequencing data," *Nature Reviews Genetics*, vol. 17, no. 8, p. 459, 2016.

[63] T. A. Lasko, J. G. Bhagwat, K. H. Zou, and L. Ohno-Machado, "The use of receiver operating characteristic curves in biomedical informatics," *Journal of Biomedical Informatics*, vol. 38, no. 5, pp. 404–415, 2005, Clinical Machine Learning, ISSN: 1532-0464. DOI: 10.1016/j.jbi.2005.02.008.

[64] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv preprint arXiv:1303.3997*, 2013.

[65] M. Kimura, T. Maruyama, and J. F. Crow, "The mutation load in small populations," *Genetics*, vol. 48, no. 10, pp. 1303–1312, Oct. 1963.

[66] N. C. Oza and K. Tumer, "Classifier ensembles: Select real-world applications," *Information Fusion*, vol. 9, no. 1, pp. 4–20, 2008.

[67] A. M. Moore, S. Patel, K. J. Forsberg, B. Wang, G. Bentley, *et al.*, "Pediatric fecal microbiota harbor diverse and novel antibiotic resistance genes," *PLOS ONE*, vol. 8, no. 11, pp. 1–9, Nov. 2013. DOI: 10.1371/journal.pone.0078822.

[68] K. J. Forsberg, S. Patel, M. K. Gibson, C. L. Lauber, R. Knight, *et al.*, "Bacterial phylogeny structures soil resistomes across habitats," *Nature*, vol. 509, no. 7502, pp. 612–616, May 2014. DOI: 10.1038/nature13377.

[69] S. R. Partridge, S. M. Kwong, N. Firth, and S. O. Jensen, "Mobile genetic elements associated with antimicrobial resistance," *Clinical Microbiology Reviews*, vol. 31, no. 4, 2018.

[70] W. A. Haynes, A. Tomczak, and P. Khatri, "Gene annotation bias impedes biomedical research," *Scientific Reports*, vol. 8, no. 1, p. 1362, Jan. 2018. DOI: 10.1038/s41598-018-19333-x.

[71] E. Smirnova, S. Huzurbazar, and F. Jafari, "PERFect: PERmutation Filtering test for microbiome data," *Biostatistics*, vol. 20, no. 4, pp. 615–631, Jun. 2018. DOI: 10.1093/biostatistics/kxy020.

[72] R. Bourgon, R. Gentleman, and W. Huber, "Independent filtering increases detection power for high-throughput experiments," *PNAS*, vol. 107, no. 21, pp. 9546–9551, 2010. DOI: 10.1073/pnas.0914005107.

[73] S. R. Eddy, "Profile hidden Markov models," *Bioinformatics*, vol. 14, no. 9, pp. 755–763, 1998.

[74] T. J. Wheeler and S. R. Eddy, "nhmmer: DNA homology search with profile HMMs," *Bioinformatics*, vol. 29, no. 19, pp. 2487–2489, 2013.

[75] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, *et al.*, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[76] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[77] W. W. Peterson, W. Peterson, E. J. Weldon, and E. J. Weldon, "Error-correcting codes," 1972.

[78] Y. Xu and T. Zhang, "Variable shortened-and-punctured reed-solomon codes for packet loss protection," *IEEE Transactions on Broadcasting*, vol. 48, no. 3, pp. 237–245, 2002. DOI: 10.1109/TBC.2002.803706.

[79] S. Burkhardt and J. Karkkainen, "Better Filtering with Gapped q-Grams," in *Combinatorial Pattern Matching*, A. Amir, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 73–85.

[80] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962. DOI: 10.1109/TIT.1962.1057683.

[81] Y. Luo, Y. W. Yu, J. Zeng, B. Berger, and J. Peng, "Metagenomic binning through low-density hashing," *Bioinformatics*, vol. 35, no. 2, pp. 219–226, Jan. 2019. DOI: 10.1093/bioinformatics/bty611.

[82] R. Britten, "Rates of dna sequence evolution differ between taxonomic groups," *Science*, vol. 231, no. 4744, pp. 1393–1398, 1986. DOI: 10.1126/science.3082006.

[83] O. Chapelle, "Training a support vector machine in the primal," *Neural Computation*, vol. 19, no. 5, pp. 1155–1178, 2007.

[84] S. Kovaka, Y. Fan, B. Ni, W. Timp, and M. C. Schatz, "Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED," *Nature Biotechnology*, Nov. 2020. DOI: 10.1038/s41587-020-0731-9.

[85] S. L. Amarasinghe, S. Su, X. Dong, L. Zappia, M. E. Ritchie, *et al.*, "Opportunities and challenges in long-read sequencing data analysis," *Genome biology*, vol. 21, no. 1, pp. 1–16, 2020.

[86] Y. Shu and J. McCauley, "GISAID: Global initiative on sharing all influenza data–from vision to reality," *Eurosurveillance*, vol. 22, no. 13, p. 30 494, 2017.

[87] G. Saunders, M. Baudis, R. Becker, S. Beltran, C. Beroud, *et al.*, "Leveraging European infrastructures to access 1 million human genomes by 2022," *Nature Reviews Genetics*, vol. 20, no. 11, pp. 693–701, 2019.

[88] R. A. Ramadhani, F. Indriani, and D. T. Nugrahadi, "Comparison of Naive Bayes smoothing methods for Twitter sentiment analysis," in *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 2016, pp. 287–292. DOI: 10.1109/ICACSIS.2016.7872720.

[89] J. D. Rodriguez, A. Perez, and J. A. Lozano, "Sensitivity Analysis of k-Fold Cross Validation in Prediction Error Estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 3, pp. 569–575, 2010. DOI: 10.1109/TPAMI.2009.187.

[90] J. J. Kozich, S. L. Westcott, N. T. Baxter, S. K. Highlander, and P. D. Schloss, "Development of a Dual-Index Sequencing Strategy and Curation Pipeline for Analyzing Amplicon Sequence Data on the MiSeq Illumina Sequencing Platform," *Applied and Environmental Microbiology*, vol. 79, no. 17, pp. 5112–5120, 2013. DOI: 10.1128/AEM.01043-13.

[91] R. C. Edgar, "Updating the 97% identity threshold for 16S ribosomal RNA OTUs," *Bioinformatics*, vol. 34, no. 14, pp. 2371–2375, Feb. 2018. DOI: 10.1093/bioinformatics/bty113.

[92] Y. Hilewitz and R. B. Lee, "Fast bit compression and expansion with parallel extract and parallel deposit instructions," in *IEEE 17th International Conference on Application-specific Systems, Architectures and Processors*, 2006, pp. 65–72. DOI: 10.1109/ASAP.2006.33.

[93] R. Stephen, "Understanding inverse document frequency: on theoretical arguments for IDF," *Journal of Documentation*, vol. 60, no. 5, pp. 503–520, Jan. 2004.

[94] A. Aizawa, "An information-theoretic perspective of tf–idf measures," *Information Processing & Management*, vol. 39, no. 1, pp. 45–65, 2003. DOI: https://doi.org/10.1016/S0306-4573(02)00021-3.

[95] R. E. Madsen, D. Kauchak, and C. Elkan, "Modeling word burstiness using the Dirichlet distribution," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 545–552.

[96] E. C. Norton and B. E. Dowd, "Log odds and the interpretation of logit models," *Health Services Research*, vol. 53, no. 2, pp. 859–878, 2018. DOI: https://doi.org/10.1111/1475-6773.12712.

[97] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[98] P. Wolfe, "Convergence conditions for ascent methods," *SIAM review*, vol. 11, no. 2, pp. 226–235, 1969.

[99] S. Wu, R. Schwartz, D. Winter, D. Conrad, and R. Cartwright, "Estimating error models for whole genome sequencing using mixtures of Dirichlet-multinomial distributions," *Bioinformatics*, vol. 33, no. 15, pp. 2322–2329, 2017.

[100] P. de Valpine and A. N. Harmon-Threatt, "General models for resource use or other compositional count data using the Dirichlet-multinomial distribution," *Ecology*, vol. 94, no. 12, pp. 2678–2687, 2013.

[101] D. D. Lewis, "Naive (Bayes) at forty: The independence assumption in information retrieval," *Machine Learning: ECML-98*, vol. 1398, 1998.

[102] N. Bouguila, "Clustering of Count Data Using Generalized Dirichlet Multinomial Distributions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 4, pp. 462–474, 2008.

[103] G. Kaur and E. N. Oberai, "A review article on Naive Bayes classifier with various smoothing techniques," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 10, pp. 864–868, 2014.

[104] J. Lilleberg, Y. Zhu, and Y. Zhang, "Support vector machines and word2vec for text classification with semantic features," in *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*, IEEE, 2015, pp. 136–140.

[105] R. D. Gupta and D. S. P. Richards, "The history of the Dirichlet and Liouville distributions," *International Statistical Review*, vol. 69, no. 3, pp. 433–446, 2001.

[106] N. Bouguila, "Count Data Modeling and Classification Using Finite Mixtures of Distributions," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 186–198, 2010.

[107] W. Fan and N. Bouguila, "Online Learning of a Dirichlet Process Mixture of Beta-Liouville Distributions Via Variational Inference," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 11, pp. 1850–1862, 2013. DOI: 10.1109/TNNLS.2013.2268461.

[108] M. Sklar, "Fast MLE computation for the Dirichlet multinomial," *arXiv preprint 1405.0099*, 2014.

[109] H. W. Eves, *Elementary matrix theory*. Courier Corporation, 1980.

[110] T. Minka, "Estimating a Dirichlet distribution," Ph.D. dissertation, Massachusetts Institute of Technology, 2000.

[111] A. Sofo, "The Basel Problem with an Extension," *Mathematical Analysis and Applications: Selected Topics*, pp. 631–659, 2018.

[112] D. Merlini, R. Sprugnoli, and M. C. Verri, "The Cauchy numbers," *Discrete Mathematics*, vol. 306, no. 16, pp. 1906–1920, 2006.

[113] F. He and X. Ding, "Improving Naive Bayes Text Classifier Using Smoothing Methods," in *Advances in Information Retrieval*, G. Amati, C. Carpineto, and G. Romango, Eds., vol. 4425, Berlin, Heidelberg: Springer, 2007.

[114] A. Cardoso-Cachopo, "Improving Methods for Single-label Text Categorization," Ph.D. dissertation, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.

[115] W. S. Cleveland, E. Grosse, and W. M. Shyu, "Statistical models in S," in *Local regression models*, T. J. Hastie, Ed., Boca Raton, FL, USA: Chapman and Hall, 1992, ch. 8.

[116] A. Cardoso-Cachopo and A. L. Oliveira, "Semi-Supervised Single-Label Text Categorization Using Centroid-Based Classifiers," in *Proceedings of the 2007 ACM Symposium on Applied Computing*, ser. SAC '07, Seoul, Korea: Association for Computing Machinery, 2007, pp. 844–851, ISBN: 1595934804. DOI: 10.1145/1244002.1244189.

[117] W. R. Gilks, G. S. Richardson, and D. J. Spiegelhalter, "Introducing Markov Chain Monte Carlo," in *Markov Chain Monte Carlo in Practice*, W. R. Gilks, G. S. Richardson, and D. J. Spiegelhalter, Eds., Boca Raton, FL, USA: Chapman and Hall, 1996, ch. 1, pp. 1–20, ISBN: 0-412-05551-1.

[118] G. Salton, *The SMART Retrieval System—Experiments in Automatic Document Processing*. USA: Prentice-Hall, Inc., 1971.

[119] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[120] R. C. Edgar, "SINTAX: a simple non-Bayesian taxonomy classifier for 16S and ITS sequences," *bioRxiv*, 2016. DOI: 10.1101/074161.

[121] R. Edgar, "Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences," *PeerJ*, vol. 9, e10805–e10805, Feb. 2021. DOI: 10.7717/peerj.10805.

[122] Y. Orenstein, D. Pellow, G. Marçais, R. Shamir, and C. Kingsford, "Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing," *PLOS Computational Biology*, vol. 13, no. 10, e1005777, 2017.

[123] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, *et al.*, "Understanding hidden memories of recurrent neural networks," in *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, IEEE, 2017, pp. 13–24.