Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

# Machine learning methods for plasmid recognition in bacterial genome assemblies

Diploma Thesis

2024
Bc. Juraj Vašut

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

# Machine learning methods for plasmid recognition in bacterial genome assemblies

Diploma Thesis

| | |
|---|---|
| Study Programme: | Computer Science |
| Field of Study: | Computer Science |
| Department: | Department of Computer Science |
| Supervisor: | doc. Mgr. Bronislava Brejová, PhD. |

Bratislava, 2024
Bc. Juraj Vašut


Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Bc. Juraj Vašut
**Študijný program:** informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
**Študijný odbor:** informatika
**Typ záverečnej práce:** diplomová
**Jazyk záverečnej práce:** anglický
**Sekundárny jazyk:** slovenský

**Názov:** Machine learning methods for plasmid recognition in bacterial genome assemblies
*Metódy strojového učenia na rozpoznávanie plazmidov v zostavených genómoch baktérií*

**Anotácia:** Plazmidy sú krátke molekuly DNA prítomné v mnohých baktériách, ktoré často prispievajú k šíreniu rezistencie na antibiotiká. Pri zostavovaní genómov z krátkych čítaní obvykle dostaneme veľký počet krátkych sekvencií nazývaných kontigy. Cieľom práce je navrhnúť metódu, ktorá určí, ktoré kontigy patria spolu do jednej molekuly DNA, ktorou môže byť plazmid alebo bakteriálny chromozóm. Základom metódy je využitie strojového učenia na klasifikáciu, či daná dvojica kontigov patrí do jednej molekuly alebo nie.

**Vedúci:** doc. Mgr. Bronislava Brejová, PhD.
**Katedra:** FMFI.KI - Katedra informatiky
**Vedúci katedry:** prof. RNDr. Martin Škoviera, PhD.

**Dátum zadania:** 16.12.2022

**Dátum schválenia:** 16.12.2022 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce


Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

## THESIS ASSIGNMENT

| | |
|---|---|
| **Name and Surname:** | Bc. Juraj Vašut |
| **Study programme:** | Computer Science (Single degree study, master II. deg., full time form) |
| **Field of Study:** | Computer Science |
| **Type of Thesis:** | Diploma Thesis |
| **Language of Thesis:** | English |
| **Secondary language:** | Slovak |

**Title:** Machine learning methods for plasmid recognition in bacterial genome assemblies

**Annotation:** Plasmids are short DNA molecules present in many bacteria, often contributing to the spread of antibiotic resistance. The result of assembling genomes from short reads is typically a large number of short sequences called contigs. The goal of the work is to design a method for determining which contigs originate from one DNA molecule, which can be a plasmid or a bacterial chromosome. The core of the method is the use of machine learning to classify whether a given pair of contigs belongs to the same molecule or not.

| | |
|---|---|
| **Supervisor:** | doc. Mgr. Bronislava Brejová, PhD. |
| **Department:** | FMFI.KI - Department of Computer Science |
| **Head of department:** | prof. RNDr. Martin Škoviera, PhD. |
| **Assigned:** | 16.12.2022 |
| **Approved:** | 16.12.2022 prof. RNDr. Rastislav Kráľovič, PhD. Guarantor of Study Programme |

.....
Student

.....
Supervisor

**Acknowledgments:** I would like to thank my supervisor doc. Mgr. Bronislava Brejová, PhD. for her helpful advice and guidance during the work on this thesis.

# Abstrakt

Plazmidy významne prispievajú k vývinu rezistencií voči antibiotikám v baktériách. Táto vlastnosť z nich robí zaujímavý cieľ výskumu. Napriek tomu zostáva problém identifikácie príslušnosti kontigov ku jednotlivým molekulám zložitý. V tejto práci predstavujeme náš prístup ku triedeniu kontigov. Tento prístup využíva informácie z grafu kontigov na klasifikáciu párov kontigov podľa príslušnosti ku rovnakej molekule a následnom zhlukovaní kontigov do jednotlivých molekúl. Výsledkom tohto prístupu sú skupiny kontigov patriacich do plazmidov alebo chromozómov.

**Kľúčové slová:** plazmid, binning, klasifikácia, zhlukovanie, bakteriálny genóm

# Abstract

Plasmids significantly contribute to an increase in antibiotic resistance. This property makes them an interesting research subject. However, the problem of identification of which molecule a contig belongs to remains difficult. In this thesis, we introduce our approach to contig binning. This approach uses information in an assembly graph to classify pairs of contigs based on if they belong to the same molecule followed by clustering of contigs into individual molecules. This approach results in groups of contigs belonging to either plasmids or chromosomes.

**Keywords:** plasmid, binning, classification, clustering, bacterial genome

# Contents

**Introduction** **1**

**1 Necessary terminology** **3**
1.1 Plasmid . . . 3
1.2 Sequencing and reads . . . 4
1.3 Genome assembly . . . 6
1.4 Contig . . . 6
1.5 Assembly graph . . . 6

**2 Existing methods for plasmid binning** **9**
2.1 MOB-recon . . . 9
2.2 Recycler . . . 10
2.3 PlasmidSPAdes . . . 11
2.4 HyAsP . . . 12

**3 Machine learning methods** **13**
3.1 Classification . . . 13
3.1.1 Logistic regession . . . 13
3.1.2 K-nearest neighbors . . . 14
3.1.3 Gaussian Naive Bayes classifier . . . 15
3.1.4 Random forest . . . 16
3.1.5 Gradient Boosting classifier . . . 17
3.2 Clustering . . . 18
3.2.1 Markov clustering . . . 18

**4 Our approach** **21**
4.1 Overview . . . 21
4.2 Data . . . 21
4.3 Classification . . . 23
4.3.1 Features . . . 23
4.3.2 Training . . . 24

4.4 Clustering . . . . . 25

**5 Experimental results** **27**

5.1 Data . . . . . 27

5.2 Training . . . . . 29

5.3 Classification . . . . . 30

5.4 Clustering . . . . . 31

**Conclusion** **35**

**Appendix A** **41**

# List of Figures


1.1 Bacterial DNA and plasmid.[23] . . . 3
1.2 Sequencing process using Illumina Miseq.[8] . . . 5
1.3 Creation of contigs from reads. . . . 6
1.4 Assembly graph.[25] . . . 7

2.1 MOB-recon workflow.[17] . . . 10
2.2 Recycler workflow.[18] . . . 11

3.1 The course of a logistic function.[22] . . . 14
3.2 K-nearest neighbors.[2] . . . 15
3.3 Random forest prediction.[24] . . . 17
3.4 Training of Gradient Boosting classifier. . . . 18
3.5 Markov clustering spliting a graph into clusters.3.5 . . . 19

4.1 Process of labeling contig pairings. (Class 0 = same molecule, Class 1 = different molecule, Class 2 = unknown) . . . 23

5.1 True distribution of data into classes. . . . 28
5.2 Distribution of pairings into classes. . . . 28
5.3 Clusters produced using different inflation represented as percentages of contigs belonging to respective classes. . . . 32
5.4 Clusters (except the largest one) produced using different inflation. . . . 32
5.5 Results of well performing clustering. . . . 33
5.6 Results of underperforming clustering. . . . 34
5.7 Composition of fragmented clusters from an assembly. . . . 34

# List of Tables

5.1 Statistics of datasets created from assemblies. For each type of pairing, contains the distribution of pairings into classes. (P = plasmid, Ch = chromosome, Other = pairings where at least 1 contig is ambiguous or unknown) . . . . . 29
5.2 Performance of trained models on test dataset. . . . . 30
5.3 Classification of datasets using different models. . . . . 31

# Introduction

Plasmids significantly contribute to an increase in antibiotic resistance. This property makes them an interesting research subject. However, the problem of identification of which molecule a contig belongs to remains difficult. In this thesis, we introduce our approach to contig binning. This approach uses information in an assembly graph to classify pairs of contigs based on if they belong to the same molecule followed by clustering of contigs into individual molecules. This approach results in groups of contigs belonging to either plasmids or chromosomes.

The resistance against antibiotics in bacteria has become a major health threat in recent years. The resistance can be spread in bacterial populations quickly with the transfer of small DNA molecules called plasmids. These molecules can be transferred horizontally in the population using the process of conjugation. As a result, the detection of plasmids in bacterial genomes is a problem with a lot of focus on it.

Despite the ability of long-read sequencing data more suited for the task of whole-genome assembly, many facilities use short reads in their whole-genome assemblies. This approach typically leads to the assemblies being fragmented, making the identification of plasmids difficult. The classification of fragments as either plasmid or chromosomal in this case does not provide the information, since the bacteria can contain multiple plasmids and chromosomes. Thus, it is important to determine from which molecule each fragment originates. This task is performed using plasmid binning.

There are several paths to plasmid binning: reference-based and de-novo. In reference-based binning, the binning method uses different reference databases that provide it with additional information such as known plasmids or plasmid genes. The reliance on databases makes these methods less capable of identification of novel plasmids. De-novo methods, on the other hand, rely on the information provided in the assembly. The binning is then based on assumptions, that the features of fragments from different molecules will vary. More recent methods combine reference-based and de-novo binning in an attempt to increase the precision.

In this thesis, we introduce a binning method using various machine learning methods to bin contigs from a bacterial assembly by utilizing an assembly graph. This method focuses on the classification of pairs of contigs and the subsequent clustering of contigs into separate bins.

In the first chapter of the thesis, we explain the terminology necessary for understanding this thesis. We describe what a plasmid is and how it differs from a chromosome. We explain the structure of an assembly graph and the processes necessary to create it from biological sequences.

In the second chapter, we focus on existing methods used in the task of binning the plasmid contigs. We explain, what information they use and how they infer the plasmid binning based on this information.

In the third chapter, we describe the machine learning methods used by our approach. We introduce both classification and clustering methods. For each method, we explain the processes used in their production of results.

In the fourth chapter, we introduce our approach to the task of plasmid binning. We provide an overview of the steps in the approach. After that, we talk about the data used and more details concerning the approach.

In the fifth chapter, we show the results of experiments performed with our approach. We discuss data preparation and the training of the classifiers used by our method. We then evaluate the results from the classification and clustering.

# Chapter 1

# Necessary terminology

In this chapter, we introduce concepts and terminology from the realm of genomics used in this thesis. We explain, what is a plasmid, and its role in the propagation of bacterial resistance. We provide a simplified explanation of what are contigs and assembly graphs and how they are created.

## 1.1 Plasmid

A plasmid is a DNA molecule, usually found in bacteria [4]. It is a molecule of DNA physically separate from the main chromosome making its replication not dependent on the replication of the chromosome (Figure 1.1). This allows for the presence of multiple copies of the same plasmid in the organism. The most common form of a plasmid is a small, circular DNA molecule with both forward and reverse strands of DNA. When compared with chromosomal DNA molecules, plasmids tend to be several times smaller.

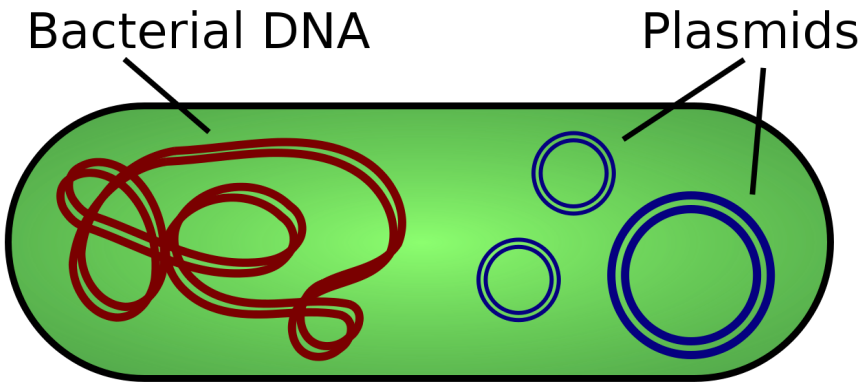


Figure 1.1: Bacterial DNA and plasmid.[23]

Another difference is that while chromosomes contain a large number of genes necessary for the correct functions of an organism, plasmids tend to contain a much smaller number of genes that are usually useful only in specific situations, and thus their presence is not required in the chromosome and the organism is capable of surviving without them. The genes that are present can for example encode information necessary to protect bacteria from antibiotics or other unusual conditions in the environment [4]. Due to their small size, plasmids can be transferred between bacteria using a process called conjugation [26]. In this process, a single strain of plasmid DNA is transferred without the necessity of reproduction of the bacteria. In both bacteria, the missing strand is then synthesized to complete the plasmid. Afterward, both bacteria are capable of spreading the plasmid further.

## 1.2 Sequencing and reads

In order to allow the analysis of DNA molecules, it is necessary to extract the genetic code from the genome into a series of reads. Reads in this context are parts of the genome that are machine- or human-readable. This readable form contains information like the order of nucleotides in the sequence as well as the quality, which represents the probability of the nucleotide being extracted correctly. The extraction is facilitated by sequencers in a process called sequencing.

There are multiple methods used to sequence DNA. Some of the most prevalently used methods are Nanopore sequencing and Next-Gen sequencing. Nanopore sequencing is implemented in Oxford Nanopore MinION [6], which produces long reads at a low cost. Its main disadvantage is the lower accuracy of the reads. Next-gen sequencing, which provides more accurate reads, is implemented in sequencers produced by Illumina, for example, Illumina MiSeq [16]. These sequencers provide short reads with high accuracy. To acquire this accuracy, the sequencing takes longer and is more costly. In this thesis, the data used is sequenced using Next-Gen sequencing providing short reads. Here, we describe the method used to create such reads. As an example, we use the process used by Illumina MiSeq.

Before starting the sequencing, the sequencer requires a DNA library (Figure 1.2a). To create a DNA library, DNA is first fragmented into small parts (500-1000 bp). After the addition of special barcoding sequences to the fragments, short oligonucleotides (adaptors) are bound to the fragments. The adaptors are complementary to primer sequences on a glass disc. The fragment is attached to the glass disc by a primer on one end, while the other is held close to another primer using its adaptor. After the attachment to the glass disc, a new strand of DNA complementary to the attached strand is synthesized. This new strand is attached to the primer the first strand is held

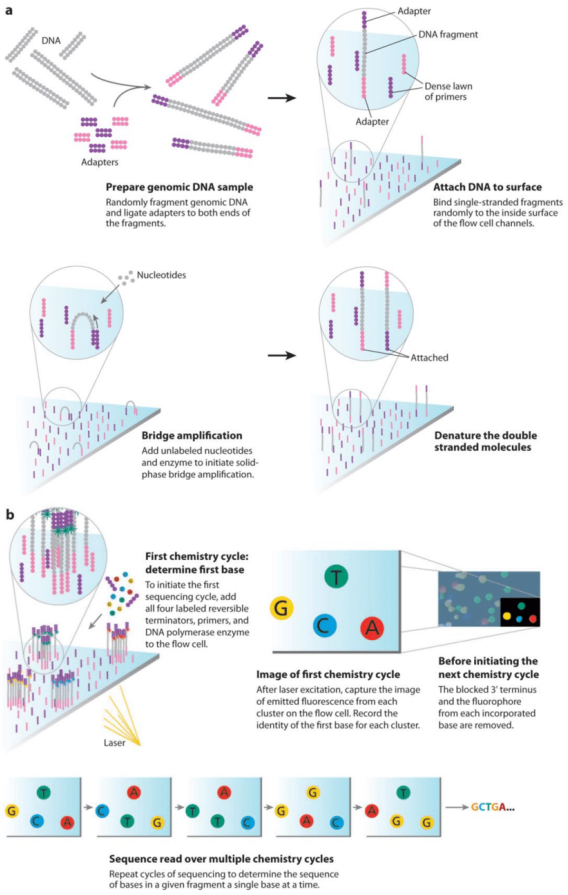


Figure 1.2: Sequencing process using Illumina Miseq.[8]

close to by its adaptor. After the synthesis, the two strands separate, and the bond between the adaptor and the primer is released. This form of replication is repeated many times, creating thousands of copies of the fragment in close proximity forming a cluster. This happens on the glass disc for every fragment of DNA, creating a DNA library.

With the library created, the sequencing can proceed (Figure 1.2b). During sequencing, a substrate called "mastermix" is used. Mastermix contains primers, DNA polymerase, and 4 different types of marked fluorescent nucleotides with a sequence inhibiting polymerization (terminators) on its 3' -end. The nucleotides are bound to the fragments of DNA based on complementarity. Next, the excess mastermix is removed from the disc. The sequencer then measures the fluorescence of the bound nucleotides to determine their identity. After the measurement, the terminators split from the sequence allowing another nucleotide to connect. This process is cyclically repeated until the entire library is read. The result is a set of reads that are used further in genome assembly.

## 1.3 Genome assembly

The goal of genome assembly is to reconnect the reads produced during sequencing into the original DNA sequences. In an ideal case, the result of the genome assembly would contain each original DNA sequence in its complete form represented as a single sequence of characters. In practice, the result of the assembly is usually more fragmented and can contain mistakes. In the case of short reads, this can be caused, for example, by long repetitive parts of the original sequence spanning longer distances than what can be bridged by the reads[6]. In such cases, the resulting assembly is comprised of smaller fragments called contigs.

## 1.4 Contig

A contig is a sequence of DNA bases assembled from reads. Contigs are created in an attempt to recreate the original sequences of DNA molecules, from which the reads are produced (Figure 1.3).


Figure 1.3: Creation of contigs from reads.

They are created by merging overlapping reads into longer contiguous sequences. In case of overlap of multiple reads on the same base, different methods are employed to choose, which base from the ones present in the reads will be present in the resulting contig. One of these methods can be a consensus, where the base is chosen as the one most frequently present among the reads.

## 1.5 Assembly graph

Representation of a genome comprised of only contigs usually does not provide all of the information contained in the reads. The information omitted can contain the relationship between contigs in the genome as well as their orientation. To capture this information, many contig assemblers provide the resulting assembly in an assembly graph rather than just pure contigs.

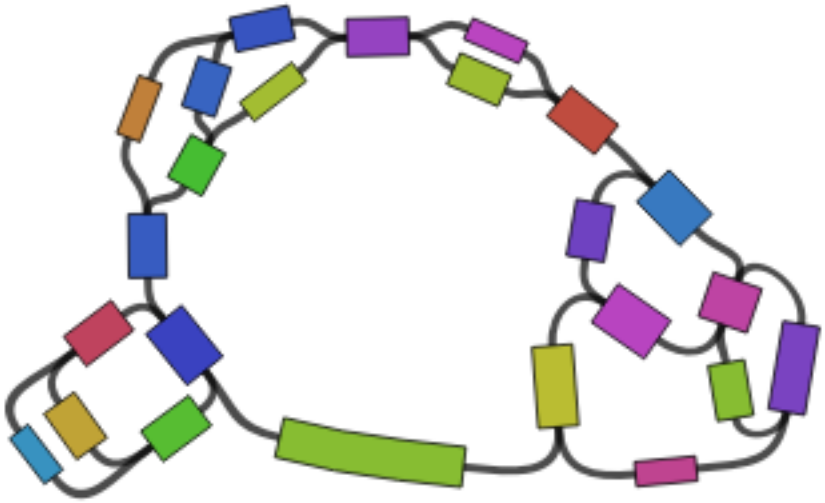

Figure 1.4: Assembly graph.[25]

In the assembly graph, each node represents a contig. These nodes have two extremities, start and end, that represent the beginning and the end of the contig. Edges between the nodes represent adjacencies between the nodes (Figure 1.4). As a result, the walks through this graph represent possible sequences present in the genome. In the walk, it is necessary that when the node is entered through one extremity, it needs to be exited through the other.

# Chapter 2

# Existing methods for plasmid binning

In this chapter, we introduce methods currently available for binning of bacterial plasmids. We explain their approach to binning and highlight deviations from our method.

Methods used in plasmid binning can be generally classified into 2 categories: reference-based and de-novo. Reference-based methods map contigs to a reference database. This allows them to bin the contigs based on the sequences they map to. Their main disadvantage is their reliance on a reference database, which makes them less reliable in binning of novel plasmids. Unlike reference-based methods, de-novo methods do not require reference sequences. As a substitute, they rely on contig features assumed to be plasmid-specific. More recent binning methods combine both reference-based and de-novo binning. Here are several methods in each category.

## 2.1 MOB-recon

MOB-recon [17] is a reference-based tool that is used to reconstruct individual plasmid sequences from genome assemblies utilizing plasmid reference databases (Figure 2.1). The tool originates from MOB-suite, a set of tools designed for clustering, reconstruction, and typing of plasmids in assemblies.

In the first step, the tool looks for contigs denoted as circular. These contigs are considered complete plasmids and are included in the result regardless of the rest of the algorithm. The algorithm then uses multiple reference databases in an attempt to identify plasmid contigs. These databases include elements that can indicate the presence of plasmids as well as repetitive elements that are not unique to plasmids as well as a database of known plasmids. The algorithm uses individual databases as queries for BLAST [9], an alignment algorithm that attempts to find an alignment of a query onto sequences in the database. In this context, the database is the assembly.

The algorithm uses the databases containing the elements indicative of plasmids to

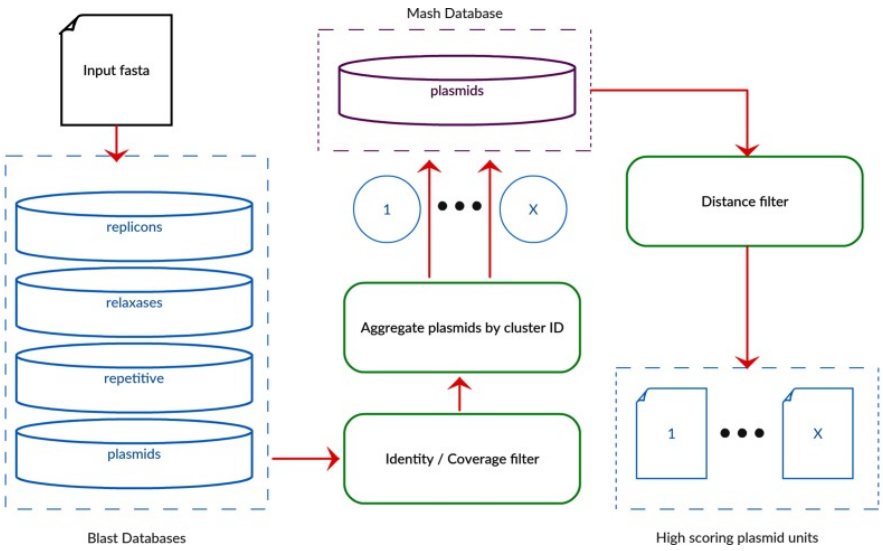


Figure 2.1: MOB-recon workflow.[17]

filter out contigs, that are likely to be chromosomal. The database containing repetitive elements is then queried against the assembly to flag potentially problematic contigs, which contain only repetitive elements and thus do not present sufficient evidence of originating from a plasmid molecule. Next, the assembly is queried against the database of known plasmids in an attempt to identify the plasmids the contigs originate from. Using the information gained from the query, the contigs are aggregated into units based on the identities of the hits in the database.

The plasmid units created are then searched for circular contigs. In the case of multiple circular contigs present in the same unit, the circular sequences are split into individual units. Units that contain contigs with only repetitive elements are then discarded. The remaining groups are considered individual plasmids. The database of plasmids is then used to identify the closest reference plasmid to each group. If the distance of the group from the reference is greater than a set threshold, the plasmid is labeled as novel.

## 2.2 Recycler

Recycler [18] uses a de-novo method to find plasmids in an assembly graph. It uses features of the sequences in the graph to determine, which contigs originate in the same plasmid. Its main focus is to produce circular groups in the graph where each group represents a single molecule. The algorithm assumes that a plasmid should have uniform coverage and few nodes. With these assumptions, the algorithm looks for cycles in the graph, that satisfy several constraints: minimum path weight for an edge, low cover variation, concordant read mapping, and sufficient sequence length. The

cycle satisfying these conditions is called a good cycle.

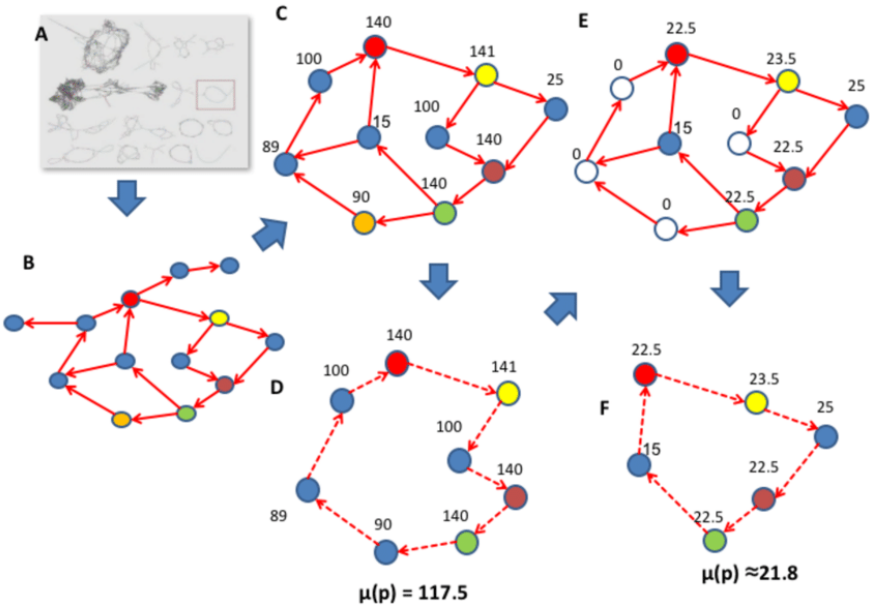


Figure 2.2: Recycler workflow.[18]

The algorithm repeatedly finds a good circle in the graph, assigns it coverage equal to the mean cycle coverage, and subtracts the coverage from the graph (Figure 2.2). Nodes, that reach a non-positive coverage are then removed from the graph. This process is repeated until no good cycle can be found. At this point, all complete plasmids are removed from the graph and the remaining nodes are put in a separate group.

## 2.3 PlasmidSPAdes

PlasmidSPAdes [1] is another de-novo method to find plasmids. It uses median coverage to differentiate between different molecules of origin. The basics of the algorithm lie in the discovery of a plasmid graph, a subgraph of an assembly graph containing only plasmid contigs.

This is done by first repeating 2 steps. It removes chromosomal edges in the assembly graph that do not belong to a connected component with no dead-end edges and is shorter than 150 kilobases. A chromosomal edge in this context is an edge with coverage that satisfies the following condition:

$$1 - maxDeviation < \frac{Coverage(e)}{medianCoverage} < 1 + maxDeviation$$

A dead-end edge is an edge for which either the number of incoming edges of the start node or the number of outgoing edges of the end node is equal to 0. If it removes at least one edge, the algorithm removes all dead-end edges. The repetition continues until no edges can be removed.

After removing the chromosomal and dead-end edges, the algorithm removes all non-plasmid components present in the assembly graph to construct a plasmid graph. To resolve potential repeats in the plasmid graph, the algorithm runs a tool exSPAnder [15]. The output contains plasmidic contigs with each assigned to a connected component in the plasmid graph.

## 2.4 HyAsP

HyAsP [11] is a hybrid algorithm that works with a database of known plasmid genes while using a greedy assembly algorithm. The algorithm is based on expanding the chains of plasmid contigs using seeds in the assembly graph. The seeds are contigs that satisfy criteria of length, the presence of known plasmid genes, and read depth. The seeds are used as starting points in the graph, from which the chains of contigs are constructed.

The seeds are enumerated based on their plasmid gene density and GC content, with a preference for high gene density and difference in GC content from the average GC content of the assembly. The greedy algorithm extends the chain by searching its endpoints for eligible extensions. Eligible extensions are contigs adjacent to the endpoints that satisfy the following criteria: length of plasmid shorter than 1750000 nucleotides, gene-free stretches shorter than 2000 nucleotides, and fluctuations in read depth and GC content lover than 15%.

The extensions are then scored using the following function:

$$score(P,B) = depth\_diff * |\frac{1 - depth(B)}{average\_depth(P)}| + gene\_density * (1 - density(B)) + gc\_diff * |gc(P) - gc(B$$

where $depth\_diff, gene\_density, gc\_diff$ are weights of individual features. The plasmid is $P$ and the contig is $B$. The algorithm then extends the plasmid using the extension with the minimal score.

If no more extensions are possible, the resulting chains are considered potential plasmids. If the first and last contigs of a chain are the same or have sufficient overlap, it is circularized. The read depth of nodes in the graph is then updated and the nodes where the read depth becomes non-zero are removed from the graph. In the postprocessing step, plasmid fragments are filtered using several criteria. The plasmids that satisfy all of the criteria are then binned using their read depth and GC content if they are not circular. Otherwise, they are output as is.

# Chapter 3

# Machine learning methods

This section showcases methods considered for individual tasks and explains the mechanisms behind them.

## 3.1 Classification

The goal of classification in our method is to determine for each pairing of contigs if they originate from the same DNA molecule. This way, we hope to differentiate both the chromosome DNA from plasmids as well as different plasmids from each other. This is done based on the assumption that contigs from different molecules have different features.

For classification, we consider several supervised machine-learning techniques. The precision of the techniques is evaluated using a confusion matrix where the goal is to achieve a high percentage of true positives and true negatives. The confusion matrix is useful in not only showing the accuracy of the trained model, but also showing which class the model has trouble predicting properly. Models considered are logistic regression, k-nearest neighbors, Gaussian Naive Bayes, Random Forest, and Gradient Boosting classifier.

### 3.1.1 Logistic regession

Logistic regression [7] is the simplest model considered. It starts with a linear combination of features. The linear combination of features is then transformed using a logistic function. The values produced by the function belong in the range between 0 and 1 (Figure 3.1). The formula of the function can be written as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where $z$ is the linear combination of features. Resulting values can then be used to represent probabilities of the data belonging to respective categories. During the

training, coefficients for input features are learned, leading to the creation of a decision boundary separating the two classes. Probabilities predicted using the trained model can either be used as is, representing the probability of the input belonging to class 1, or used to split data into classes deterministically using a chosen threshold as the point of split.

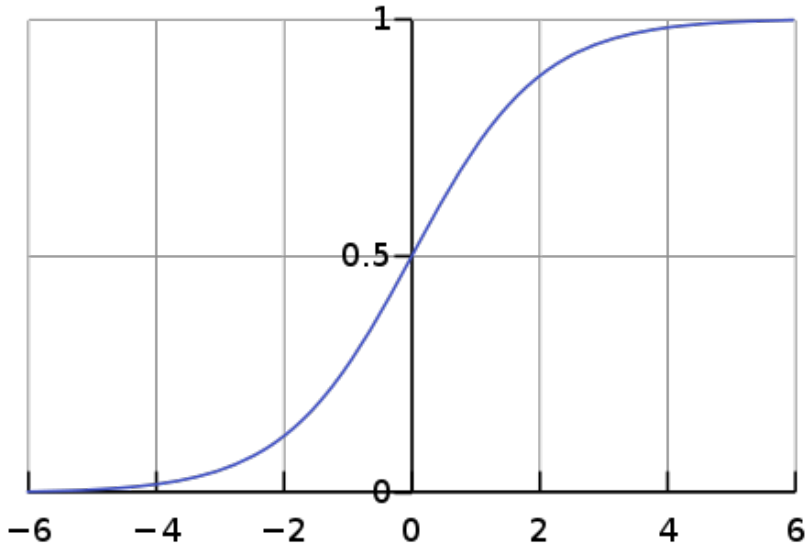


Figure 3.1: The course of a logistic function.[22]

### 3.1.2 K-nearest neighbors

The basic idea behind the k-nearest neighbors [14] algorithm is that similar data points tend to belong to the same class. This means that the data point is likely to belong to the same class as its neighbors. Unlike many other algorithms, the k-nearest neighbors algorithm does not use its training phase to optimize any function. Instead, the available data points and their class labels are stored and then used directly in the predictions. During predicting, the algorithm simply observes $k$ nearest data points from the training set (Figure 3.2). The class of the majority of observed neighbors is then assigned to the data point that needs to be classified. The distance between the data points is determined using a distance metric, in our case Euclidian distance. The formula to calculate the distance is calculated as:

$$\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

where $p$ and $q$ are data points and $n$ is the number of features. In this algorithm, it is important to choose the value of $k$. With a smaller value, the model becomes more

sensitive to noise in the data, which can lead to overfitting. A high value can lead to a more robust model not influenced by noise with the downside of overlooking local patterns in the data.

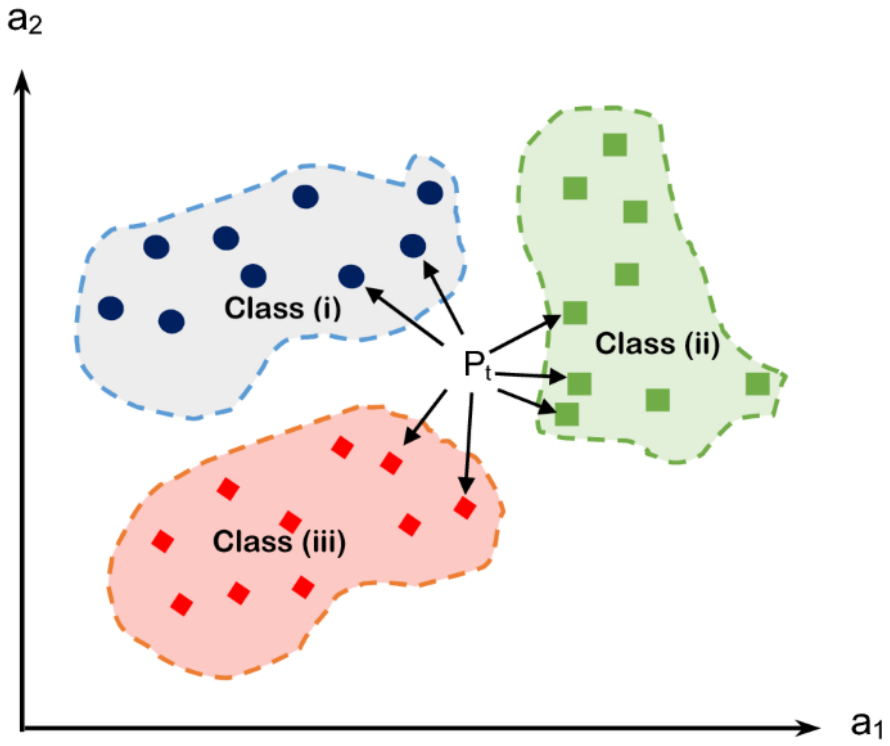


Figure 3.2: K-nearest neighbors.[2]

### 3.1.3 Gaussian Naive Bayes classifier

Gaussian Naive Bayes classifier works with an assumption that the features of a data point are conditionally independent given the class. This means that the presence of a feature in a class is unrelated to the presence of any other feature. The core of the algorithm is Bayes theorem, which describes the probability of a hypothesis given some evidence. Its formula is mathematically represented as:

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k) \cdot P(C_k)}{P(\mathbf{x})}$$

where $P(C_k|\mathbf{x})$ is the posterior probability of class $C_k$ given the observation $\mathbf{x}$, $P(\mathbf{x}|C_k)$ is the likelihood of observing $\mathbf{x}$ given class $C_k$, $P(C_k)$ is the prior probability of class $C_k$ and $P(\mathbf{x})$ is the evidence of marginal likelihood. The algorithm also assumes that the continuous-valued features follow a normal distribution. This simplifies the computation of the likelihood, as the probability density function of normal distribution is

well-known, described as:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

where $x$ is the value of the random variable, $\mu$ is the mean of the distribution and $\sigma^2$ is the variance. During the training phase, the model estimates the mean and variance of each feature for each class in the training dataset. When a data point is being classified, the model computes the posterior probability of each class. The class with the highest probability is then assigned to the data point.

### 3.1.4 Random forest

Random forest [12] is a model that combines multiple individual models called decision trees in order to produce its predictions. This model, by virtue of using the results of multiple models, tends to be robust against overfitting. A decision tree is a hierarchical structure that splits the feature space into regions recursively, with each region corresponding to a particular prediction. Each split attempts to use a single feature that splits the data into the cleanest subsets possible. Random forest introduces randomness into the decision trees in two forms during the training phase. Each tree is provided only a random subset of the training dataset, resulting in a diversity between the trees. The tree also considers only a random subset of features for each split, further enhancing the diversity of the trees as well as preventing them from becoming overfitted to the training data.

During the prediction, each tree in the Random forest independently makes a class prediction. The final prediction is then determined by aggregating the predictions and producing the result. This is achieved by a majority vote among the trees (Figure 3.3). While the model provides accurate predictions, it has a big disadvantage when it comes to the time of training due to the number of models that need to be trained. For the same reason, it is also a model that takes up the most space. To minimize these disadvantages, it is important to optimize the number of trees in the forest as well as their maximum depth and the number of features considered for each split.

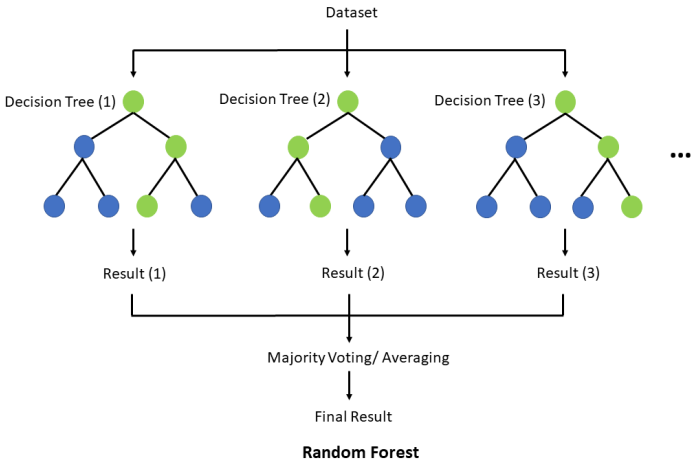


Figure 3.3: Random forest prediction.[24]

### 3.1.5 Gradient Boosting classifier

Similarly to the Random Forest, Gradient Boosting classifier [3] is also often based on training decision trees. Unlike Random Forest, Gradient Boosting classifier builds decision trees sequentially, based on the errors of the previous tree. These trees tend to be shallow. Each tree is fit to the negative gradient of the loss function with respect to the prediction of the trees built so far (Figure 3.4). In this way, the model learns to approximate the gradient of the loss function with respect to the current prediction. Trees are trained until the model runs out of the number of trees available or until no improvements in the performance of the model are observed. The loss function used by the model is binary cross-entropy, which can be described as:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

where $N$ is the number of samples, $y_i$ is the true label of the sample and $\hat{y}_i$ is the predicted probability that the sample belongs to class 1. The loss function is minimized using gradient descent. By controlling the learning rate of the gradient descent, the contribution of each distribution tree in the model can be manipulated. With a lower learning rate, the model becomes more robust to overfitting at the cost of the number of iterations required for the model to converge. To predict classification, Gradient Boosting combines the predictions of all decision trees in the model weighted by their contribution to the final prediction. The class with the highest probability is then selected as the final prediction.

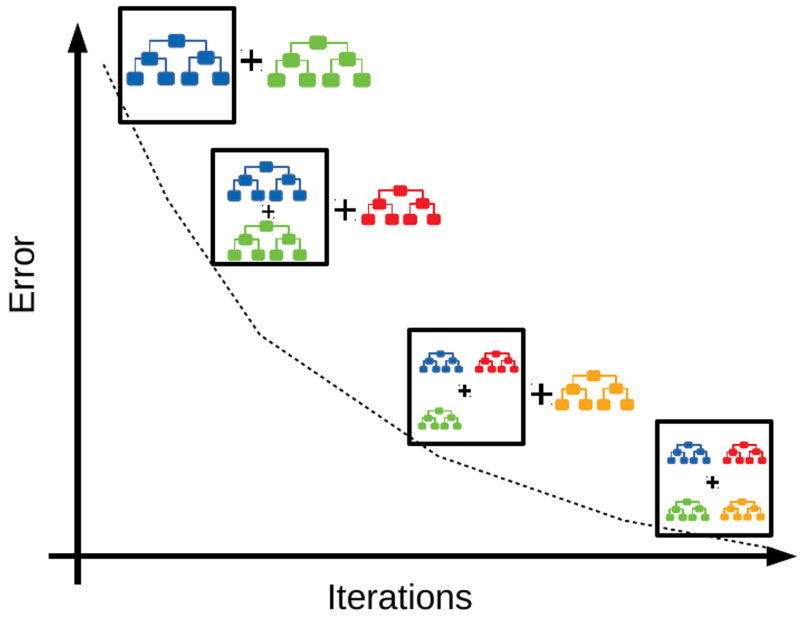


Figure 3.4: Training of Gradient Boosting classifier.

## 3.2 Clustering

The goal of clustering in our implementation is to group contigs that originate from the same molecule. In the ideal conditions, each cluster would contain all of the contigs from exactly 1 molecule. Although not ideal, we also permit clusters not containing every contig of the molecule. More important is to split the contigs in a way that puts only contigs from a single molecule in the cluster.

### 3.2.1 Markov clustering

Markov clustering [20] is an unsupervised machine learning method designed for performing clustering on graphs. It is an effective way of identifying clusters within large graphs. The algorithm operates on a graph where nodes represent data points, in our case individual contigs, and edges represent a relationship between them. Relationships in our graph are indicative of if we consider the contigs to be from the same molecule. The input of this algorithm is an adjacency matrix representing the graph.

The matrix is first converted into a stochastic matrix. The stochastic matrix is a square matrix where each column sums up to 1. In this context, the matrix represents probabilities of transitioning from one node to another in a random walk across the graph. From this point, the algorithm consists of 2 repeating steps.

The first step is expansion. In this step, the algorithm simulates a random walk through the graph by performing successive matrix multiplications. This process is used to emphasize the paths that can be easily reached from a given node. Each

expansion step takes the $k$th power of the current stochastic matrix.

This step is followed by inflation. In the inflation step, each element in the matrix is raised to a power of $r$. Afterward, the matrix is again scaled so that each column once again sums up to 1. As a result of the inflation, strong connections in the graph are amplified while weak connections are further weakened.

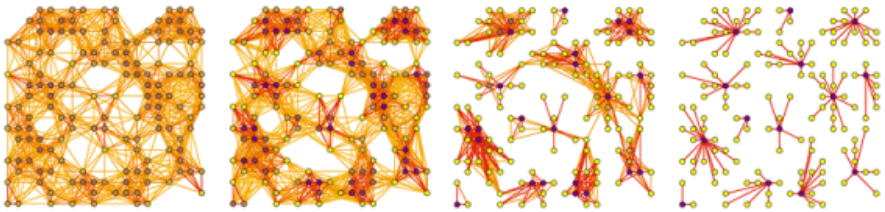

Figure 3.5: Markov clustering spliting a graph into clusters.3.5

The algorithm alternates between these steps until it converges. In reality, the process converges very fast towards a limit that is invariant under both expansion and inflation, where the convergence around the limit points is quadratic. In the process, inflation uncouples the nodes, splitting the graph (Figure 3.5). This in turn creates components in the graph, clustering its nodes. At the point of convergence, the resulting components are output as the resulting clusters.

# Chapter 4

# Our approach

In this chapter, we explain our approach to the task of plasmid binning. We discuss the data used in our experiments, the approach to their processing, and the features we extract. We also introduce libraries used in our approach and the parameters used in their methods.

## 4.1 Overview

The goal of our approach is to group contigs based on which molecule they originate from. The method we implement consists of 2 steps: classification and clustering. In classification, our goal is to decide for pairs of contigs if they should be considered as belonging to the same molecule. To this end, we train classifiers mentioned in Chapter 3. These classifiers are trained on data that contains correct classifications for the pairs.

The results of the classification can then be used in the second step, clustering. By clustering the contigs based on the classification of the pairs, we attempt to separate the contigs into groups that represent individual molecules. Markov clustering used in our approach is explained in Chapter 3. The classification of pairings is used as a distance matrix in a graph where the weight of an edge corresponds to the probability that the contigs connected by it belong to the same molecule. The rest of this chapter explains the details of our approach.

## 4.2 Data

For each bacterial sample, we have contigs and an assembly graph created from short reads, an assembly created from both short and long reads (called a hybrid assembly), alignment of contigs from short reads to the hybrid assembly, and whether the contig in hybrid assembly is circular or not. The short read assembly is used as

the input for the training of the classifier or as the input for the task of binning. The hybrid assembly along with the mapping and circularity information is used to infer the correct labeling of the data from the short assemblies for training and testing purposes. Additionally, we also use annotation of contigs of the short assembly determining if the contig is chromosomal, plasmid, or ambiguous. These labels are provided in results from plASgraph2[19], a tool designed to detect plasmid contigs from an assembly graph. During the preprocessing, we combine the data in the following way.

To facilitate easier manipulation with the assembly graph, we employ python library *networkx* [5]. This library contains methods to work with the graphs. By observing the nodes in the graph, we extract the features of contigs. We also include the classification of whether the contig is from plasmid, chromosome, or ambiguous.

After preparation of the features, we move on to filter some contigs out from the dataset. These contigs include those shorter than 100 base pairs since these contigs can contain a high amount of unnecessary noise. Because we want to focus on the classification of plasmids, we also attempt to remove contigs that are chromosomal. This removal is based on the plasmid and chromosomal scores, representing the probability of the contig originating from the respective molecule, present in the result of plASgraph2. We first find candidate contigs for removal. These are the contigs that have a chromosomal score higher than 0.8, meaning they are likely of chromosomal origin.

We then look at the neighbors of the candidate contigs in the assembly graph. If all of the neighbors have a chromosomal score higher than 0.8, we remove the candidate contig from the graph. Otherwise, the candidate contig is kept in the graph. This is done in order to retain connections between plasmid and chromosomal contigs while removing chromosomal contigs that are not necessary for our goal. The contigs remaining in the graph retain the features from the whole graph. This way, we only remove long chromosomal contigs that would otherwise skew the results of the classification.

Using the remaining contigs, we create our training dataset. This dataset contains pairings between all of the contigs. To provide labels for this dataset, we look at the types of contigs in the hybrid assembly, which we use as a reference. The labels we assign split, the dataset into 3 classes. We also use the results from plASgraph2 to assist the classification. In cases, where the contigs are determined to be from different types of molecules (plasmid and chromosome) based on the labels provided by plASgraph2, we label them as such. In other cases, the labeling is done using the hybrid assembly (Figure 4.1).

First, inspect the sets of reference contigs that each of the contigs in the pairing maps to. If these sets overlap, it means that there exists a contig in the reference that both of the contigs map to. In this case, the contigs are deemed as from the same molecule.

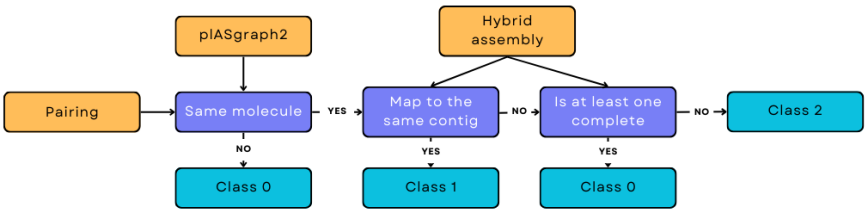


Figure 4.1: Process of labeling contig pairings. (Class 0 = same molecule, Class 1 = different molecule, Class 2 = unknown)

If they do not map to the same reference contig, we look at the completeness of the reference contigs. We consider a reference contig as complete if it is circular. This is due to the nature of DNA molecules in bacteria, which tend to be circular. Otherwise, we consider the contig incomplete. In cases where at least one of the contigs is mapped only to complete reference contigs, that means that the contigs are not from the same molecule and are labeled as such.

Finally, if none of the conditions mentioned is satisfied, it means that it is impossible to determine from the provided information if the contigs are from the same molecule. This is due to the fact that they do not appear on the same reference contigs. At the same time, it is possible that there exists a connection between two incomplete reference contigs that the paired contigs did map to. These pairings are removed from the dataset as we do not know the correct answer needed for the training of the classifier. The remaining pairings are used in the training of the classifier.

## 4.3 Classification

The goal of classification in our method is to determine for each pairing of contigs if they originate from the same DNA molecule. This way, we hope to differentiate both the chromosome DNA from plasmids as well as different plasmids from each other. This is done based on the assumption that contigs from different molecules have different features.

### 4.3.1 Features

We characterize each contig using features that are usually deemed to be indicative of which molecule the contig belongs to [**?**]. These features include length, coverage, GC content, degree of a node, and frequencies of k-mers present.

The length of the contig can mainly be used to determine if the contig is from chromosomal DNA or a plasmid. This is because contigs from chromosomal DNA tend

to be longer than plasmid contigs due to chromosomal DNA molecules being longer [19]. We transformed the actual length using a natural logarithm.

Another feature is coverage. Coverage denotes how many reads, on average, cover each base of the contig. This feature can be considered one of the most important for achieving the task since different molecules are likely to have different coverage. At the same time, the contigs from the same molecule are expected to have similar coverage.

GC content denotes the percentage of bases in the contig that are either guanine or cytosine. Similarly to the coverage, GC content also tends to be different in different molecules. This makes it an important indicator in determining where the contig originates from. As a feature, the GC content of a contig is normalized by subtracting the GC content of the entire assembly. Under this condition, the GC content of chromosomal contigs is closer to the total GC content due to the chromosome encompassing a large portion of the total assembly. Conversely, plasmid contigs can be further from the total GC content.

The next feature used is the k-mer profile of the contig. K-mers are short sequences of length $k$ present in the contig. In our case, we set $k$ to 5. A k-mer profile describes the frequency distribution of possible k-mers in the contig. As in plASgraph2, we use similarity between the k-mer profile of the contig and the entire assembly [19]. This feature, along with the GC content is often used in plasmid detection [19].

Finally, the degree of a node is the number of edges leaving the node in the assembly graph. In other words, it is the number of connections between the contig and other contigs in the assembly.

The features mentioned represent individual contigs. Since our goal in the classification is to discover, if a pair of contigs originates from the same molecule, each data point contains features from both contigs. Additionally, each pairing contains combined features that describe the differences in the contig features. We also include the distance between the contigs in the assembly graph. In case there is no direct path between the contigs, the distance is set to 100000 in place of the absent information. This is done in order to provide more features describing the relation between the contigs. For each pairing, we also include the distance (the number of nodes) between paired contigs in the assembly graph.

### 4.3.2 Training

The pairings created during the preprocessing were used to train the models in several combinations of features. The first combination included only features from individual contigs and their distance in the assembly graph, totaling 11 features. In the results section, we denote this combination of features as combination 0. Another combination included only the combined features, meaning features such as the difference between

the contig lengths. It also included the distance of contigs in the assembly graph, resulting in a total of 6 features. This combination is referred to in the results as combination 1. The last configuration of features was a combination of all of the features, making it the most comprehensive with 16 features. This final combination is denoted in the results as combination 2.

We trained classifiers for every combination mentioned. To make sure that all of the results are replicable, we set a discrete seed for classifiers that can use some degree of randomness. The machine learning methods used in our approach are available in $scikit - learn$ [13] library and include the following: $LogisticRegression$, $GaussianNB$, $GradientBoostingClassifier$, $KNeighborsClassifier$, and $RandomForestClassi$ Aside from setting a discrete seed where applicable, we use these models in their default setup.

For verification of results, we created a test dataset comprising 10% data points. Since the resulting training dataset did not contain the same number of positive and negative class samples, we randomly sampled the dataset in order to avoid possible bias towards the class with the higher number of samples. The sampling used was undersampling since creating new data by introducing noise could introduce incorrect data points which corresponding class would be impossible to determine.

In our approach we use a random undersampler implemented in the *imblearn* [10] library. As with the models themselves, we set a fixed seed for the undersampler to make the results replicable. By undersampling the dataset, we further decreased its size, which could lead to a less capable classifier. For this reason, we trained classifiers both with balanced and original datasets. The trained models are saved in binary format using the *pickle* [21] library. This way, the models do not need to be retrained with each use of our method.

## 4.4 Clustering

For clustering, we used the implementation of Markov clustering available in the *markov_clustering* python library. This implementation requires an adjacency matrix as an input. We create this matrix by first creating a graph from contigs present in the assembly using the *networkx* library. If contigs are classified as being from the same molecule, we add an edge between them in the graph. In our approach, we attempt 2 different variations of classification. We use either discrete classes or the probability of the pair originating in the same molecule.

As a result, the shape of the graph is also different. In the case of using the probability, every node in an assembly is connected by an edge in the resulting graph with the weight of an edge being equal to the probability determined by the classification. On

the other hand, with the use of discrete classes, the edges are not weighted and are only present between the contigs that are determined to be from the same molecule by our classifier. We extract the adjacency matrix from the graph using the *to_numpy_array* function present in the *networkx* library.

The resulting adjacency matrix is then used in the algorithm to produce the final clustering. We use the algorithm with the inflation set to 3.0, which we have determined to be the optimal setting. We output the resulting clusters as the individual bins of contigs representing molecules.

# Chapter 5

# Experimental results

In this chapter, we present the results of our experiments. We provide statistics regarding the dataset used in the training of the classifier as well as data used in our experiments. We provide commentary on the results of experiments and evaluate them.

## 5.1 Data

The data used in our experiments is the data used in the development of plASgraph2. This data contains 70 assemblies from 8 species of bacteria. After the preprocessing, we were left with 3 datasets: complete, balanced, and testing. Out of these, the balanced dataset is used in training the classifiers and the test dataset is used to measure their precision. We include the complete dataset for comparison.

The complete dataset contained a total of 1933791 pairings, 673039 of which were labeled as not from the same molecule. Since there was an imbalance between the classes, we used the balanced dataset, which was produced by the undersampling. This dataset contained 1346078 pairs with the split between classes being $1 : 1$. Finally, the test dataset contained 214866 pairings. This dataset was not resampled and contained 74491 pairs of contigs not belonging to the same molecule. The distribution of classes is visualized in Figure 5.1.

In Figure 5.2, we can also observe the distributions of different types of pairings in the datasets. Here, we can see that pairings of plasmid to plasmid are observed with approximately the same frequency between classes. On the other hand, pairings between chromosomal contigs are more likely to be from the same molecule. This is due to the fact that bacteria usually contain only a single chromosome. The distribution of pairings between a plasmid and a chromosomal contig shows that all of those pairings are not from the same molecule, which makes sense since plasmid and chromosome are different molecules.

Aside from the datasets used in the training of the classifier and its testing, we

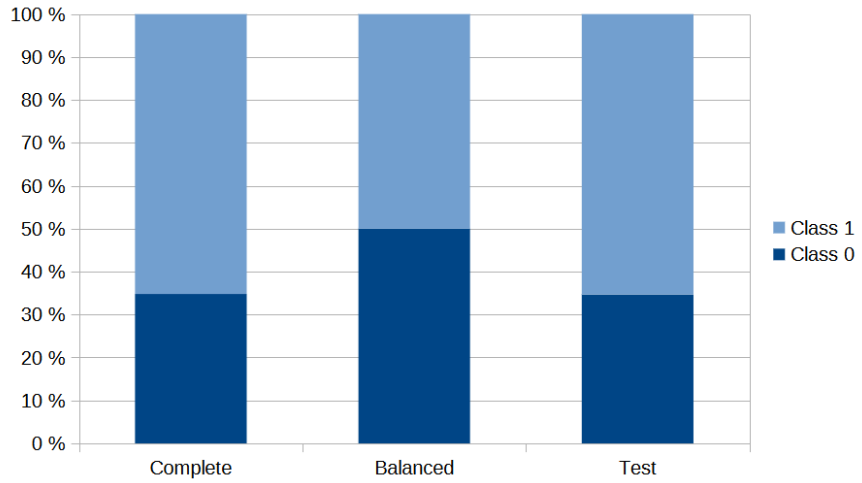


Figure 5.1: True distribution of data into classes.

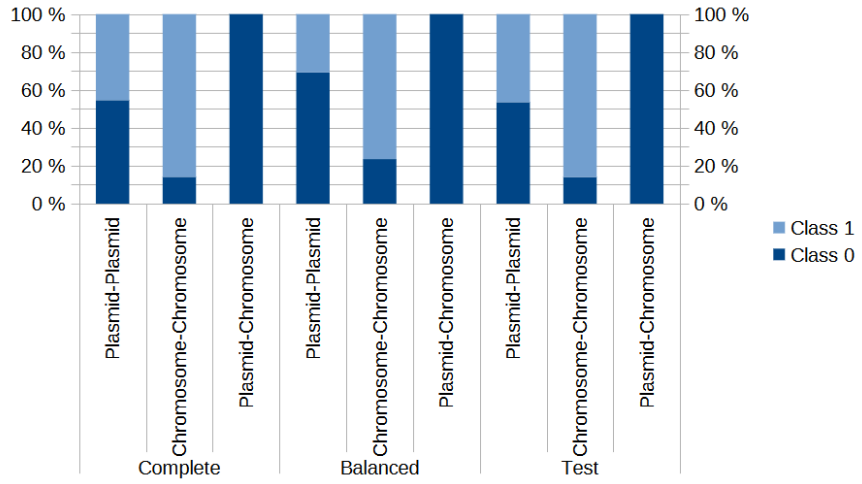


Figure 5.2: Distribution of pairings into classes.

also used 2 assemblies from each of the 8 species of bacteria to test the capability of our method to bin the contigs. These assemblies were not balanced to provide real-life test scenarios. In the statistics of the assemblies used in experiments (Table 5.1), we observed that several assemblies contained 0 pairings of chromosomal contigs classified as not from the same contig. We consider this an ideal situation as it indicates that the entire chromosome in the assembly is properly connected. If the number of pairings is greater than 0, it suggests that the hybrid assembly used as a reference contains either multiple chromosomal contigs or unlabeled contigs that contain sections that are deemed chromosomal. This can be caused by either the bacteria containing multiple chromosomal molecules or an incomplete assembly.

Another interesting observation comes in pairings of plasmids. Here, we can observe in several assemblies an expected number of plasmids. In particular, in cases where all

| Organism | Assembly | Size | Ch-Ch | | P-P | | P-Ch | | Other | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Class 1 | Class 0 | Class 1 | Class 0 | Class 1 | Class 0 | Class 1 | Class 0 |
| *Acinetobacter baumannii* | abau-SAMEA12292493 | 31709 | 20349 | 2645 | 450 | 145 | 0 | 7525 | 450 | 145 |
| | abau-SAMEA12292546 | 30775 | 15400 | 0 | 0 | 0 | 0 | 176 | 4930 | 10269 |
| *Staphylococcus aureus* | saur-SAMD00180477 | 7396 | 4656 | 0 | 1 | 2 | 0 | 291 | 1626 | 820 |
| | saur-SAMD00258402 | 9834 | 6441 | 0 | 15 | 11 | 0 | 912 | 1301 | 1154 |
| *Escherichia coli* | ecol-SAMN15148412 | 14509 | 4465 | 0 | 3 | 0 | 0 | 285 | 3842 | 5914 |
| | ecol-SAMN15148693 | 13041 | 4656 | 97 | 0 | 1 | 0 | 196 | 2505 | 5586 |
| *Klebsiella pneumoniae* | kpne-SAMN21366037 | 73528 | 4429 | 2012 | 2973 | 3697 | 0 | 13224 | 10765 | 36428 |
| | kpne-SAMN15148557 | 89411 | 6786 | 117 | 4825 | 5752 | 0 | 20886 | 16336 | 34709 |
| *Pseudomonas aeruginosa* | paer-SAMEA5578860 | 10940 | 1122 | 849 | 72 | 299 | 0 | 5220 | 2194 | 1184 |
| | paer-SAMEA12292475 | 55939 | 41328 | 288 | 21 | 0 | 0 | 2023 | 12248 | 31 |
| *Enterobacter cloacae* | eclo-SAMN15148612 | 6439 | 2017 | 0 | 24 | 21 | 0 | 660 | 3038 | 679 |
| | eclo-SAMN15148668 | 25191 | 3916 | 179 | 885 | 1195 | 0 | 5915 | 2729 | 10372 |
| *Enterobacter spp.* | exxx-SAMN15148756 | 26920 | 9593 | 137 | 236 | 64 | 0 | 3500 | 3938 | 9452 |
| | exxx-SAMN15148503 | 24573 | 7260 | 0 | 334 | 101 | 0 | 3630 | 3874 | 9374 |
| *Enterococcus feacalis* | efae-E8481 | 12922 | 2600 | 640 | 128 | 148 | 0 | 1944 | 2128 | 5334 |
| | efae-E7441 | 33102 | 4899 | 2979 | 971 | 1307 | 0 | 8636 | 7564 | 6746 |

Table 5.1: Statistics of datasets created from assemblies. For each type of pairing, contains the distribution of pairings into classes. (P = plasmid, Ch = chromosome, Other = pairings where at least 1 contig is ambiguous or unknown)

of the pairs are classified as from the same molecule, we can assume that there is only a single plasmid present in the assembly. In one case, there is a single pairing between plasmids and it is classified as not from the same molecule. Here we can assume that there are 2 plasmids present in the assembly. In one other assembly, there are no pairings between plasmid contigs. There are, however, some pairings between plasmid and chromosomal contigs. From this, we can infer that there is a single, potentially complete, plasmid present in the assembly.

The assemblies also contain a large number of contigs that were not classified by plASgraph2, resulting in a large number of pairings where we cannot say, what kind of pairing it is. Despite this, we were still able to assign them labels based on the information present in the reference assemblies.

## 5.2 Training

For the task of classification, we trained every model mentioned in the machine learning methods with all of the combinations of features, resulting in a total of 15 models. Here, we measure their performance (Table 5.2).

Logistic regression had a very bad performance. It achieved the best performance with the feature combination 1. Even in this case, it had very low accuracy in predicting the negative class. Its prediction of a positive class was also one of the worst.

The Gaussian Naive Bayes classifier had by far the lowest precision. It classified most of the pairs as positive regardless of whether they were positive or not. Different feature combinations did not provide significantly different results.

Along with logistic regression, the Gaussian Naive Bayes classifier was deemed as not fit for our task due to its poor results. On the other hand, K-nearest neighbors, Random Forest, and Gradient Boosting classifiers performed much better. All of them

| Model | Combination type | Size (kB) | Accuracy | True Positive | False Negative | False Positive | True Negative |
|---|---|---|---|---|---|---|---|
| Logistic regression | 0 | 1 | 0,63205 | 119726 | 20649 | 58412 | 16079 |
| | 1 | 1 | 0,63786 | 121015 | 19360 | 58452 | 16039 |
| | 2 | 1 | 0,63416 | 120455 | 19920 | 58687 | 15804 |
| Gaussian Naive Bayes | 0 | 1 | 0,66524 | 135432 | 4943 | 66985 | 7506 |
| | 1 | 1 | 0,66396 | 135322 | 5053 | 67150 | 7341 |
| | 2 | 2 | 0,66548 | 135584 | 4791 | 67085 | 7406 |
| K-nearest neighbors | 0 | 150024 | 0,85289 | 119173 | 21202 | 10407 | 64084 |
| | 1 | 92323 | 0,64175 | 91257 | 49188 | 27858 | 46633 |
| | 2 | 178777 | 0,84345 | 117917 | 22458 | 11180 | 63311 |
| Random forest | 0 | 1284314 | 0,96234 | 134242 | 6133 | 1958 | 72533 |
| | 1 | 4227336 | 0,69236 | 98355 | 42020 | 24082 | 50409 |
| | 2 | 1263241 | 0,95441 | 133181 | 7194 | 2602 | 71889 |
| Gradient Boosting | 0 | 133 | 0,78039 | 111409 | 28966 | 18221 | 56270 |
| | 1 | 133 | 0,66626 | 92210 | 48165 | 23545 | 50946 |
| | 2 | 133 | 0,78227 | 111741 | 28634 | 18149 | 56342 |

Table 5.2: Performance of trained models on test dataset.

performed the worst with feature combination 1, with Random Forest suffering a loss in accuracy of more than 20%. The classifier also more than tripled in size.

With other feature combinations, the performance of the 3 models was not significantly influenced, with the precision only slightly decreased with the use of feature combination 2 along with an increase in the size of the models. As a result, we deemed the best set of features to be combination 0.

The best performance by far can be observed in the Random Forest classifier, with the accuracy of classification reaching over 96%. This was combined with good precision. In the classification of the assemblies, we used the 3 well-performing classifiers.

## 5.3 Classification

For each of the 3 classifiers used we provide results of classification (Table 5.3) on the 16 datasets selected from the total of 70 assemblies available.

Once again, the Random Forest classifier tends to perform better on all datasets. We can also observe that the models performed similarly on the individual assemblies and the test dataset. However, in some cases, the models generate more false positives than true negatives, suggesting that the models might be overtrained on the positive class. This can lead to potential problems in clustering.

| Organism | Assembly | Model | True Positive | False Negative | False Positive | True Negative |
|---|---|---|---|---|---|---|
| *Acinetobacter baumannii* | abau-SAMEA12292493 | Random Forest | 27672 | 4054 | 13348 | 19339 |
| | | K-nearest neighbors | 20422 | 11304 | 14450 | 18237 |
| | | Gradient Boosting | 18196 | 13530 | 11979 | 20708 |
| | abau-SAMEA12292546 | Random Forest | 21042 | 1093 | 7852 | 2603 |
| | | K-nearest neighbors | 15315 | 6820 | 5625 | 4830 |
| | | Gradient Boosting | 14315 | 7820 | 4532 | 5923 |
| *Staphylococcus aureus* | saur-SAMD00180477 | Random Forest | 6326 | 56 | 743 | 371 |
| | | K-nearest neighbors | 5249 | 1133 | 474 | 640 |
| | | Gradient Boosting | 5876 | 506 | 830 | 284 |
| | saur-SAMD00258402 | Random Forest | 7890 | 108 | 1050 | 1031 |
| | | K-nearest neighbors | 6241 | 1757 | 736 | 1345 |
| | | Gradient Boosting | 7474 | 524 | 1626 | 455 |
| *Escherichia coli* | ecol-SAMN15148412 | Random Forest | 8152 | 355 | 5468 | 731 |
| | | K-nearest neighbors | 5866 | 2641 | 3970 | 2229 |
| | | Gradient Boosting | 5886 | 2621 | 3530 | 2669 |
| | ecol-SAMN15148693 | Random Forest | 6919 | 242 | 5203 | 677 |
| | | K-nearest neighbors | 4940 | 2221 | 3459 | 2421 |
| | | Gradient Boosting | 5319 | 1842 | 3293 | 2587 |
| *Klebsiella pneumoniae* | kpne-SAMN21366037 | Random Forest | 17294 | 5921 | 19779 | 41317 |
| | | K-nearest neighbors | 14346 | 8869 | 31225 | 29871 |
| | | Gradient Boosting | 10376 | 12839 | 19376 | 41720 |
| | kpne-SAMN15148557 | Random Forest | 25498 | 8744 | 15100 | 50767 |
| | | K-nearest neighbors | 18155 | 16087 | 24102 | 41765 |
| | | Gradient Boosting | 11962 | 22280 | 14002 | 51865 |
| *Pseudomonas aeruginosa* | paer-SAMEA5578860 | Random Forest | 2267 | 1214 | 368 | 7271 |
| | | K-nearest neighbors | 2389 | 1092 | 4127 | 3512 |
| | | Gradient Boosting | 1687 | 1794 | 2481 | 5158 |
| | paer-SAMEA12292475 | Random Forest | 58918 | 3317 | 63 | 2307 |
| | | K-nearest neighbors | 43354 | 18881 | 1377 | 993 |
| | | Gradient Boosting | 41756 | 20479 | 1198 | 1172 |
| *Enterobacter cloacae* | eclo-SAMN15148612 | Random Forest | 4546 | 612 | 62 | 1368 |
| | | K-nearest neighbors | 3602 | 1556 | 560 | 870 |
| | | Gradient Boosting | 3251 | 1907 | 436 | 994 |
| | eclo-SAMN15148668 | Random Forest | 6995 | 1509 | 6655 | 11637 |
| | | K-nearest neighbors | 6035 | 2469 | 9093 | 9199 |
| | | Gradient Boosting | 5294 | 3210 | 6505 | 11787 |
| *Enterobacter spp.* | exxx-SAMN15148756 | Random Forest | 13400 | 1527 | 7413 | 6101 |
| | | K-nearest neighbors | 10186 | 4741 | 6596 | 6918 |
| | | Gradient Boosting | 9706 | 5221 | 5274 | 8240 |
| | exxx-SAMN15148503 | Random Forest | 11119 | 1119 | 7277 | 6136 |
| | | K-nearest neighbors | 8775 | 3463 | 7060 | 6353 |
| | | Gradient Boosting | 7887 | 4351 | 5555 | 7858 |
| *Enterococcus feacalis* | efae-E8481 | Random Forest | 5052 | 975 | 3226 | 5112 |
| | | K-nearest neighbors | 4035 | 1992 | 4193 | 4145 |
| | | Gradient Boosting | 3793 | 2234 | 2706 | 5632 |
| | efae-E7441 | Random Forest | 15174 | 4140 | 4237 | 17852 |
| | | K-nearest neighbors | 12465 | 6849 | 9836 | 12253 |
| | | Gradient Boosting | 10269 | 9045 | 6140 | 15949 |

Table 5.3: Classification of datasets using different models.

## 5.4 Clustering

We run the clustering only for the Random Forest classifier as we have already shown that it provides the most reliable results (Table 5.3). For clustering, we tested different values of the inflation parameter. This allows us to manipulate the sensitivity of the algorithm. For the sake of compactness, we only include results from different values of inflation for one assembly, $abau - SAMEA12292493$.

With the clustering based on the probability of contigs belonging to the same molecule, we observed that the increase in inflation did not lead to improvement in clustering. The resulting clustering contained either 1 or 2 clusters providing no useful information. This is likely due to the number of edges present in the graph being very

large, making it more difficult for the algorithm to converge into an optimal solution. As a result, we do not use the probability of contigs belonging to the same molecule in the remaining experiments. Instead, we use clustering based on the discrete classes.

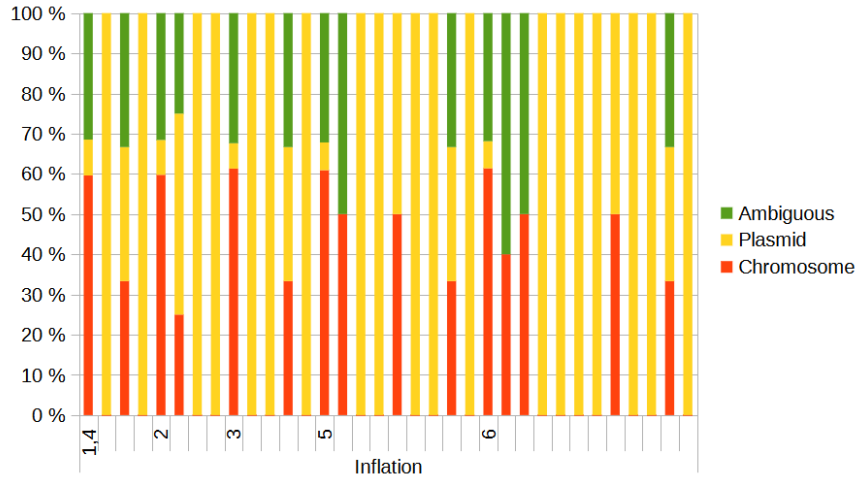


Figure 5.3: Clusters produced using different inflation represented as percentages of contigs belonging to respective classes.

In the case of clustering based on the discrete classes, we can observe a rise in the number of clusters with the increase in the value of inflation (Figure 5.3). In Figure 5.3, each column represents a single cluster. Each value of inflation on the x-axis begins a new set of clusters. This aligns with our understanding that higher inflation leads to a larger number of clusters. However, the increase in the number of clusters is not indicative of an increase in precision, as can be seen when looking at the actual number of contigs separated from the main cluster (Figure 5.4).

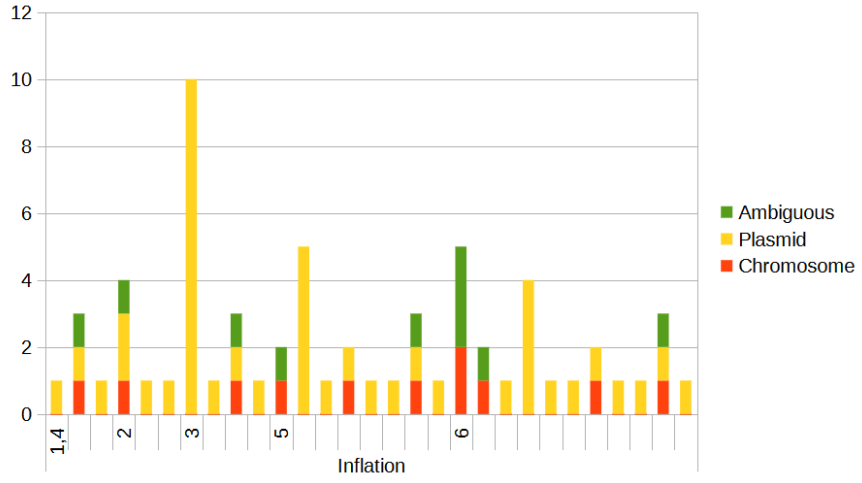


Figure 5.4: Clusters (except the largest one) produced using different inflation.

With the inflation higher than 3, the clusters start to get more fragmented with chromosomal contigs being removed from the main cluster, which contains mostly chromosomal contigs. These contigs are then inserted into clusters containing plasmids. These clusters get fragmented as well. Since the inflation value of 3 performed well, we use it in the clustering of other assemblies. In Figures 5.5, 5.6, and 5.7, we illustrate the results of clustering. In these pictures, the height of a bar represents the number of contigs of a specific type present in a cluster. Each group of 3 bars then represents an individual cluster, with clusters belonging to each assembly being to the left of their name on the x-axis.

With the inflation of 3, results on the assemblies varied. In 5 assemblies, the clustering performed well with most of the plasmids separated into clusters different than the chromosomal (Figure 5.5). This suggests that for these the inflation was set correctly.

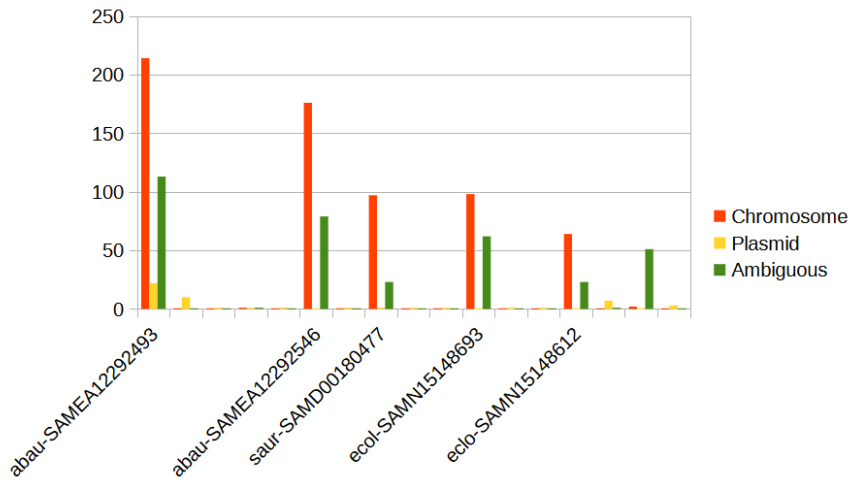


Figure 5.5: Results of well performing clustering.

9 of the assemblies either did not get separated into clusters or contained few clusters with a small number of plasmids (Figure 5.6). In the case of these assemblies, a higher inflation is probably required to achieve better results.

Finally, 2 of the assemblies resulted in a large number of fragmented clusters (Figure 5.7). For these assemblies, a lower inflation might be necessary. In Figure 5.7, we can see that the first and the second clusters separate chromosomal from plasmid contigs well. Other clusters, on the other hand, contain only a small number of contigs with some chromosomal contigs mixed in. The clusters denoted fragmented are actually a large number of clusters containing only 1 contig each merged together to make the figure more compact.

Overall, the results suggest that the inflation variable should not be fixed and should rather be optimized for each assembly individually.

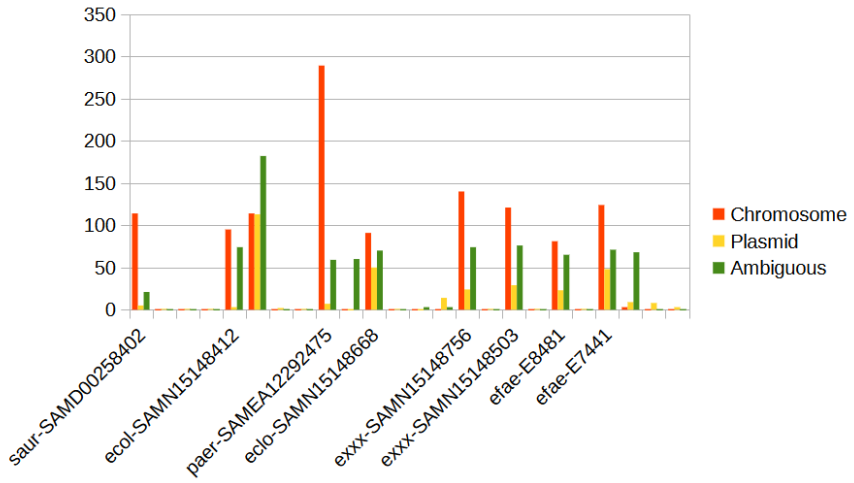


Figure 5.6: Results of underperforming clustering.

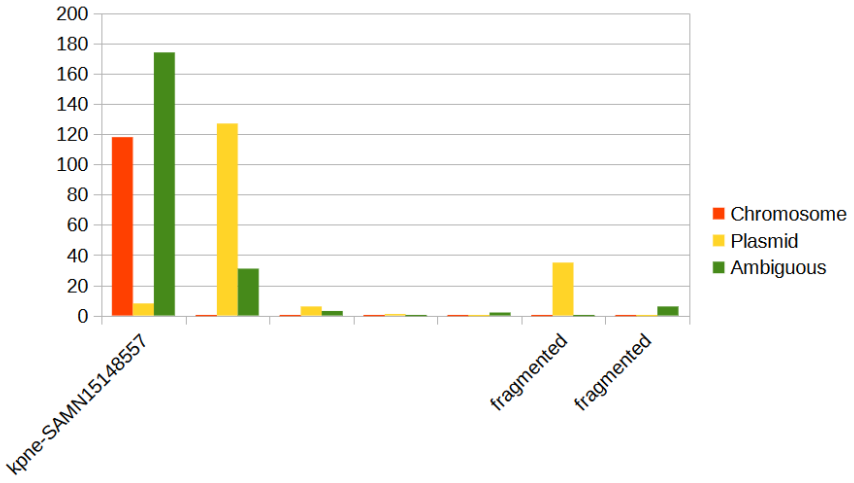


Figure 5.7: Composition of fragmented clusters from an assembly.

# Conclusion

In this thesis, we have implemented a binning method using the classification of contig pairings and clustering. In the classification step, we have shown that the best classifier for the task was the Random Forest classifier. Other than that, K-nearest neighbors and Gradient boosting classifiers performed well. The remaining classifiers performed poorly.

We have also shown that the Markov clustering algorithm performs better with the input created from discretly classified pairings. The results of clustering have shown that with a fixed value of inflation, the clusters produced will vary in quality depending on the assembly. This means that optimization of inflation may be necessary for each assembly before clustering.

In order to improve the results of binning, there is room for modifications. We propose several modifications to different parts of the method.

The first modification is the implementation of clustering that would take into consideration small distances between the contigs inferred from our classification. Even though the clustering algorithm used in our implementation does work on distance matrices of a graph, it does not appear to be sensitive enough to work on graphs with a large number of vertices of similar distances. This makes it difficult to differentiate between individual clusters. To address the issue, it is possible to modify the sensitivity of the clustering algorithm using the inflation parameter. However, as we have shown in our experiments, this allows miss-clustering of contigs and makes the fine-tuning of the sensitivity difficult and largely dependent on the input. In the case of the use of different clustering algorithms, we believe that the classification performed by our implementation along with the clustering could provide usable binning.

Next, the introduction of different features may increase the quality of classification. Although the classification provided using our implementation is of good quality, further improvement could be introduced using other features. As an example, distance in the graph could be better represented as distance in base pairs instead of distance of nodes used in our implementation. Another possible modification could include distinguishing between the ends of contigs in the graph, which would also modify the distance metric. It would also be capable of better reflecting the information provided in the assembly graph.

Another possible modification is the usage of different classification algorithms. In our implementation, we tested the performance of 5 classification algorithms, some of which were not necessarily suitable for this task. The use of larger and more complex models may be capable of better representation of the input data thus allowing more accuracy in the classification. These could include methods such as support vector machines, which can be useful in classification tasks with many features. In the same area, the increase in the size of the training dataset would likely contribute to the accuracy of the predictions.

Additionally, the removal of more chromosomal contigs could remove some bias in the process of training. Since we are more interested in distinguishing between different plasmids rather than between chromosomal and plasmid contigs, the possible decrease in overall accuracy would be outweighed by an increase in the capability to distinguish between individual plasmids.

With these modifications, we believe that our binning method could provide results on par with other plasmid binning methods.

# Bibliography


[1] Dmitry Antipov, Nolan Hartwick, Max Shen, Mikhail Raiko, Alla Lapidus, and Pavel A Pevzner. plasmidspades: assembling plasmids from whole genome sequencing data. *Bioinformatics*, 32(22):3380–3387, 2016.

[2] Dalia Atallah, Mohammed Badawy, and Ayman El-Sayed. Intelligent feature selection with modified k-nearest neighbor for kidney transplantation prediction. *SN Applied Sciences*, 1, 10 2019.

[3] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54:1937–1967, 2021.

[4] Judith E Bouma and Richard E Lenski. Evolution of a bacteria/plasmid association. *Nature*, 335(6188):351–352, 1988.

[5] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[6] Miten Jain, Hugh E Olsen, Benedict Paten, and Mark Akeson. The oxford nanopore minion: delivery of nanopore sequencing to the genomics community. *Genome biology*, 17:1–11, 2016.

[7] David G Kleinbaum, K Dietz, M Gail, Mitchel Klein, and Mitchell Klein. *Logistic regression*. Springer, 2002.

[8] Kamila Knapik. *Genetic analysis of bacteriophages from clinical and environmental samples*. PhD thesis, 07 2013.

[9] Ian Korf, Mark Yandell, and Joseph Bedell. *Blast*. " O'Reilly Media, Inc.", 2003.

[10] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[11] Robert Müller and Cedric Chauve. Hyasp, a greedy tool for plasmids identification. *Bioinformatics*, 35(21):4436–4439, 2019.

[12] Aakash Parmar, Rakesh Katariya, and Vatsal Patel. A review on random forest: An ensemble classifier. In *International conference on intelligent data communication technologies and internet of things (ICICI) 2018*, pages 758–763. Springer, 2019.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[15] Andrey D Prjibelski, Irina Vasilinetc, Anton Bankevich, Alexey Gurevich, Tatiana Krivosheeva, Sergey Nurk, Son Pham, Anton Korobeynikov, Alla Lapidus, and Pavel A Pevzner. Exspander: a universal repeat resolver for dna fragment assembly. *Bioinformatics*, 30(12):i293–i301, 2014.

[16] Rupesh Kanchi Ravi, Kendra Walton, and Mahdieh Khosroheidari. Miseq: a next generation sequencing platform for genomic analysis. *Disease gene identification*, pages 223–232, 2018.

[17] James Robertson and John HE Nash. Mob-suite: software tools for clustering, reconstruction and typing of plasmids from draft assemblies. *Microbial genomics*, 4(8):e000206, 2018.

[18] Roye Rozov, Aya Brown Kav, David Bogumil, Naama Shterzer, Eran Halperin, Itzhak Mizrahi, and Ron Shamir. Recycler: an algorithm for detecting plasmids from de novo assembly graphs. *Bioinformatics*, 33(4):475–482, 2017.

[19] Janik Sielemann, Katharina Sielemann, Broňa Brejová, Tomáš Vinař, and Cedric Chauve. plasgraph2: using graph neural networks to detect plasmid contigs from an assembly graph. *Frontiers in Microbiology*, 14:1267695, 2023.

[20] Stijn Van Dongen. Graph clustering via a discrete uncoupling process. *SIAM Journal on Matrix Analysis and Applications*, 30(1):121–141, 2008.

[21] Guido Van Rossum. *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020.

[22] Qef via Wikimedia Commons. The logistic sigmoid function., 2008.

[23] Spaully via Wikimedia Commons. Bacterium with its chromosomal dna and several plasmids, 2007.

[24] TseKiChun via Wikimedia Commons. Random forest explain, 2021.

[25] Ryan R Wick, Mark B Schultz, Justin Zobel, and Kathryn E Holt. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, 31(20):3350–3352, 2015.

[26] N Willetts and B Wilkins. Processing of plasmid dna during bacterial conjugation. *Microbiological reviews*, 48(1):24–41, 1984.

# Appendix A: Implementation

This thesis includes an electronic attachment containing the source code of our implementation and models trained.

The scripts are in the main directory. File `data_maker.py` contains functions used to create features from assemblies. File `class_trainer.py` contains functions used to train classifiers. File `classification.py` contains functions used to classify datasets. File `clustering.py` contains functions used for clustering of the contigs.

The main directory also includes files `single.txt` and `testing.txt`. These files contain paths to the samples used in our experiments as testing. File `single.txt` contains the path to the sample used in testing of different values of inflation.

In the `data` directory, we include multiple directories containing samples used in our experiments. It also includes the file `plasgraph2.csv` containing results from the tool plASgraph2.

In the directory `models`, we include a trained Gradient Boosting classifier. This classifier is not the best performing one. The best performing Random Forest classifier is not included due to the size constraints.

We also include a `README.txt` file, which includes usage and explanations for parameters of functions implemented in the Python files. It also includes a set of commands that can be used to perform binning using the pre-trained model and training of a new model.