Norwegian University
of Life Sciences

Master's Thesis 2024 60 ECTS
Faculty of Chemistry, Biotechnology and Food Science

Using machine learning for source attribution of *Listeria monocytogenes*

Terese Ryan Andersen
Bioinformatics and Applied Statistics

Acknowledgements

This thesis is a part of the master's degree in Bioinformatics and Applied Statistics at the Faculty of Chemistry, Biotechnology, and Food Science (KBM) at the Norwegian University of Life Science (NMBU). The study conducted for this master's thesis was done in collaboration with the section for Epidemiology at the Norwegian Veterinary Institute.

I would like to thank my supervisor Karin Lagesen at the Norwegian Veterinary Institute. I am grateful for all the guidance and knowledge you have shared with me, and for all the meaningful discussions and help you have provided throughout this work.

I would also like to thank my supervisor Lars-Gustav Snipen at the Norwegian University of Life Science for all the feedback you have given me and helping me to improve my work.

Lastly, I would like to thank my family and friends for the support throughout this year, especially my boyfriend Jonas Antonsen for always believing in me.

Ås, May 2024

Terese Ryan Andersen

Abstract

In this study source attribution was combined with machine learning for the purposes of making models that can predict the source of new cases of infection caused by *Listeria monocytogenes*. *L. monocytogenes* causes the infection listeriosis in humans, and the main source of infection is through food. Although it is considered a low pathogenic bacterium, the mortality rate for infected humans makes it a public health issue. Listeriosis is particularly dangerous for the elderly, the immune suppressed and for pregnant individuals. A quick identification of the source of infection is key to stopping further spread of the bacteria. By using genomic data from *L. monocytogenes* with known sources, a machine learning model may be trained to classify the bacterial isolates by sources. The trained model can then predict the sources of new cases. The available information in the genomic data was also explored to investigate if it was diverse enough to be used for partitioning isolates by source. Machine learning has already shown potential for source attribution of *L. monocytogenes* and other pathogens in studies from other countries. The origin of the data set in this study was Norwegian and contained data of whole genome sequenced *L. monocytogenes* isolates. The possibility of separating the isolates and being able to predict the sources utilizing the genetic information were explored with different kinds of machine learning methods, representation of the genomes, and subsets of the genes in the data set. The results of this research suggest that allelic profiles from both core genome and whole genome Multi Locus Sequence Typing (MLST) methods gives input data that are diverse enough for machine learning models to use for source attribution. The machine learning method Random Forest could use the allelic profiles directly as input data and had good predicting performance for the isolates with food-associated sources in this study but had poorer performance for the isolates with not food-associated sources. The method Support Vector Machine needed scaling of the input data to predict well and had similar predicting performance as the Random Forest method. The last machine learning method in this study was a neural network which was the method with the highest use of time and computational resources. The neural network performed poorer than the other methods but showed potential and better predicting performance can possibly be obtained with more tuning to improve the model.

Sammendrag

I denne studien er smittesporing kombinert med maskinlæring for å kunne lage modeller som kan predikere smittekilden til nye infeksjonstilfeller forårsaket av *Listeria monocytogenes*. Infeksjonen listeriose hos mennesker er forårsaket av *L. monocytogenes*, og den hyppigste smittemåten er gjennom mat. Selv om den regnes som en lavpatogen bakterie gjør den høye dødeligheten for infiserte personer den til et folkehelseproblem. Listeriose er spesielt farlig for eldre, immunsupprimerte og gravide personer. En rask identifikasjon av smittekilden er viktig for å stanse spredningen av bakterien. Ved å bruke data fra genomene til *L. monocytogenes* hvor kilden er kjent kan en maskinlæringsmodell trenes til å klassifisere bakterieisolater etter kilde. Den trente modellen kan videre brukes til å predikere smittekilden til nye tilfeller. Informasjonen som finnes i dataen fra genomene ble også utforsket for å se om den inneholder nok variasjon til å kunne brukes til å skille isolater etter kilde. Maskinlæring har allerede vist potensiale for bruk ved smittesporing av *L. monocytogenes* og andre patogener i studier fra andre land. Datasettet i denne studien er innhentet fra Norge og inneholder helgenomsekvenserte isolater av *L. monocytogenes*. For å utforske muligheten til å skille isolatene og predikere smittekilden ved å bruke data fra genomer ble forskjellige typer maskinlæringsmodeller, representasjon av genomene, og mindre grupper av gener fra datasettet utforsket. Resultatene fra denne forskingen tyder på at allel-profiler fra både kjernegenom og helgenom Multi Locus Sequence Typing (MLST) metoder gir nok variasjon i dataene til at maskinlæringsmodeller kan bruke den til smittesporing. Maskinlæringsmetoden Random Forest brukte allel-profilene direkte som inputdata og hadde gode prediksjonsevner for isolater fra mat-assosierte kilder, men dårligere for isolater fra ikke mat-assosierte kilder. Metoden Support Vector Machine trengte skalering av inputdataene for å kunne få bedre prediksjonsevner, og fikk omtrent de samme prediksjonsevnene som Random Forest. Den siste maskinlæringsmetoden som ble benyttet i denne studien var et neuralt nettverk, og dette var den metoden med størst bruk av tid og data-resurser. Det neurale nettverket hadde dårligere prediksjonsevner enn de andre metodene, men viste potensial og bedre prediksjonsevner kan antagelig oppnås ved mer tilpasning av modellen.

Table of contents

Abbreviations

BLAST	Basic Local Alignment Tool
BSR	BLAST scoring ratio
CC	Clonal complex
CDS	Coding sequence
cgMLST	Core genome Multi Locus Sequence Typing
DNA	Deoxyribonucleic acid
L. monocytogenes	*Listeria monocytogenes*
MLST	Multi Locus Sequence Typing
PFGE	Pulse-Field Gel Electrophoreses
rbf	Radial basis function
ReLU	Rectified Linear Unit
RFLP	Restriction Fragment Length Polymorphism
ST	Sequence type
wgMLST	Whole genome Multi Locus Sequence Typing

1.0 Introduction

1.1 Listeriosis

The bacterium *Listeria monocytogenes* is the cause of the infection listeriosis in humans. The bacterium is low pathogenic, and infection is relatively rare (Degré et al., 2010, pp. 223-226). For people with a healthy immune system listerioses is often mild and passes without treatment, but for immune suppressed, elderly, infants, or pregnant individuals listeriosis can be fatal (Degré et al., 2010, pp. 223-226). Most of the severe cases involve patients above 60 years old (Degré et al., 2010, pp. 223-226). Listeriosis can cause sepsis and other severe infections in humans, and it can cause miscarriages and stillbirth for pregnant individuals or severe infections in the newborn (Degré et al., 2010, pp. 223-226). The total mortality rate for clinical cases of listeriosis is around 25% (Ditlefsen & Egeland, 2023). Because of the high mortality rate, it is important to rapidly find the source of listeriosis outbreaks to prevent the spread to persons with high risk of severe infections.

Listeriosis is a foodborne infection and its main path of introduction to humans is through the food supply chain (Degré et al., 2010, pp. 223-226; NicAogáin & O'Byrne, 2016). *L. monocytogenes* is found in a range of different surroundings like soil and water, agricultural environments, certain animals, and in food processing facilities (Degré et al., 2010, pp. 223-226; Liao et al., 2023). Because of *L. monocytogenes'* ability to survive in high stress environments it is a pathogen that is hard to eradicate. It endures pH and temperature changes, high salt concentration and other conditions that normally reduce bacterial survival (Liao et al., 2023; NicAogáin & O'Byrne, 2016). *L. monocytogenes* can grow in refrigerator temperatures, as it has the ability to multiply in temperatures down to 0 °C (Degré et al., 2010, pp. 223-226). Because it is common to find in food processing facilities, strict regulations on the concentration of the bacteria in food products are enforced in Europe (NicAogáin & O'Byrne, 2016). High temperatures kills *L. monocytogenes*, so food products that are cooked or heat treated before consumption are rarely a source for human infection (NicAogáin & O'Byrne, 2016). Poorly washed vegetables, raw or undercooked meat and fish, unpasteurised dairy products, and meats meant to be consumed cold are common sources for listerioses (Degré et al., 2010, pp. 223-226; Folkehelseinstituttet, 2023).

In Norway the annual number of reported listeriosis cases is low, between 15-40 cases (Folkehelseinstituttet, 2023). Most of the cases of listeriosis in Norway have their origins within the country (Lyngstad et al., 2022). All severe listeriosis cases are logged in a reporting system. In sporadic cases the source is normally not investigated, however, when several cases occur within a certain time span, an outbreak investigation is launched (Folkehelseinstituttet, 2023). The Norwegian Institute of Public health (NIPH) follows an outbreak guide that contains a procedure for formulating a hypothesis for infection source (Kapperud, 2018). In this procedure there is an extensive search where all available information about the outbreak is gathered. A lot of this information is gathered by interviewing patients about their food consumption before the infection, and testing samples from potential sources and patients (Kapperud, 2018). The interviewing is time consuming and rely on people's ability to remember what food they have consumed and where they got it from. People might misremember or not be able to participate in these enquiries because of their condition. Sampling potential food sources can also cause difficulties because there might not be any of the food product left for testing, or traces of the bacteria may be hard to detect. When bacterial isolates are found in potential sources part of confirming the true source of an outbreak is comparing bacterial isolates sampled from the patients and the source using epidemiological markers like the genetic material of the bacteria (Kapperud, 2018). Sometimes it is not possible to get a sample or find isolates in suspected sources, and the source has to be inferred by calculation of likelihood based on the information gathered during the search (Kapperud, 2018). By implementing a machine learning model to make prediction for potential sources early in the search process the range of suspected sources may possibly be narrowed faster and the need for sampling can be decreased. The machine learning model may predict likely sources for the outbreak based only on the information in the genetic material of the bacteria in the clinical samples. This can also be helpful in the cases where there is not an available isolate from the source to confirm, and the likelihood must be estimated using information gathered in the search.

1.2 Genomes and genotyping

The genome of an organism contains all the hereditary genetic information of the organism (Klug et al., 2013, pp. 1-14). The genetic information is contained in molecular structures called deoxyribonucleic acid (DNA), which are comprised of nucleotides. There are 4 nucleotides in DNA, adenine, guanin, thymine, and cytosine, and their sequence order in the

DNA encodes the genetic information of organisms (Klug et al., 2013, pp. 1-14). The part of these DNA sequences that encodes functional elements like proteins are called genes, and the location of each gene sequence in the DNA is referred to as its locus (Klug et al., 2013, pp. 1-14). The characteristics of an organism are encoded by its genes. These characteristics and functions are often referred to as the organism's phenotype (Klug et al., 2013, pp. 1-14). The same gene can be encoded in different forms known as the alleles of that gene, and differences in alleles may give rise to differences in the phenotype the gene expresses. The different allele types found in an organism are the organism's genotype (Klug et al., 2013, pp. 1-14).

Genetic diversity within a species arises from diversity in the genomes and genes (Klug et al., 2013, pp. 1-14). Different mechanisms are behind the occurrence of genetic diversity within a species. For bacteria the flow of the genetic information can be either vertical between bacteria of different generation through cell division, or horizontal between bacteria when bacteria transfer part of their DNA to other bacteria (Tortora et al., 2004, pp. 210-243). When new DNA is transferred into a bacterium's cell caused by horizontal transfer a recombination of the recipient bacterium's DNA can occur and new alleles and genes can be introduced (Tortora et al., 2004, pp. 210-243). Changes in a bacterium's DNA can also occur because of mutations and mobile elements in the bacterium's DNA (Tortora et al., 2004, pp. 210-243). Mutations are small, random substitutions between the nucleotides making up the DNA sequence (Tortora et al., 2004, pp. 210-243). Mutations can create new alleles, or they can destroy genes and make them inactive. Mobile elements are small parts of the DNA sequences that can change its place along the sequence within one bacterium's genome, or these elements can be part of the horizontal transfer between bacteria of the same generation (Tortora et al., 2004, pp. 210-243). Like mutations these changes can introduce new alleles or destroy genes. These changes caused by recombination, mutations and mobile elements gives rise to genetic diversity in a bacteria species. Natural selection then drives the selection for survival of the fittest bacterial isolates (Tortora et al., 2004, pp. 210-243). Bacteria with genetic adaptations giving enhanced chances of survival will be the ones with the best opportunities to multiply and replicate their genomes to new generations. The kind of phenotypes that will be advantageous depends on the environment and other factors impacting the bacteria in their surroundings (Tortora et al., 2004, pp. 210-243).

Genetic diversity within a bacteria species can be exploited to divide bacteria into subtypes. When diversity is caused by adaptation to an environment the bacterial subtype with the adaptations can be associated with this environment. Further, knowing the subtype of a bacterial isolate, when there is an association between subtypes and environment, can be used to indicate the source of the isolate. Microbial subtyping in general tries to find characteristics in the data collected from bacterial isolates to divide the species into subtypes (Pires et al., 2009). Genotyping is when genomic data is used for microbial subtyping (Pires et al., 2009). When characteristics such as biological or biochemical qualities are used for subtyping it is known as phenotyping (Pires et al., 2009).

Genotyping can be done in several different ways depending on the genomic data available for the bacterial isolates. For many years genotyping methods using Restriction Fragment Length Polymorphism (RFLP) analysed with Pulse-Field Gel Electrophoreses (PFGE) was one of the most used methods (Martin et al., 1998; Nemoy et al., 2005). This method uses enzymes to fragment the isolate's genome at specific locations, giving fragments of different lengths (Graves & Swaminathan, 2001). The fragments are then analysed using electrophoresis, and the patterns the fragments generate during migration over the electrophoreses gel creates a PFGE profile. The PFGE profiles can be used to divide the species into subtypes (Graves & Swaminathan, 2001). This typing method gained popularity because it could be standardised between laboratories (Graves & Swaminathan, 2001). Today, methods using DNA sequencing data are often used. The increasing access to and easy sharing of sequencing data between laboratories are some of the reasons the sequence-based methods are becoming popular (Nemoy et al., 2005). Multi Locus Sequencing Type (MLST) methods are one of these methods and it uses sequencing data to find the allele variation between isolates genomes within a species (Martin et al., 1998). The MLST methods use schemas containing the loci of specific genes of interest and an allelic profile is made for the genes in the schema (Maiden et al., 2013). The profiles are made by giving distinct allele sequences at each gene a unique label in the form of a number, creating an allelic profile of the combination of numbers (Martin et al., 1998). Further, bacterial isolates with the same profile gets the same sequence type (ST) number which can be used for subtyping (Martin et al., 1998). An example to illustrate this using 4 bacterial isolates and a MLST schema for 7 loci is shown in Table 1. The numbers for each locus in the schema constitute the allelic profiles for the isolates, and identical profiles get the same sequence type number.

Table 1. Example of the MLST method for genotyping of bacterial isolates. A schema of 7 loci is used to make allelic profiles for 4 bacterial isolates and to divide them into sequence types based on similarity in the allelic profiles. The allelic profile for each isolate is the combination of the number values assigned to their allele type found at each locus.

Isolate no.	**Locus 1**	**Locus 2**	**Locus 3**	**Locus 4**	**Locus 5**	**Locus 6**	**Locus 7**	**Sequence type**
1	3	7	1	4	1	5	3	**1**
2	5	3	1	3	3	4	2	**2**
3	3	7	1	4	1	5	3	**1**
4	4	2	9	9	5	2	1	**3**

Different schemas can be made containing different loci depending on the genes of interest. MLST usually has a schema of 7 loci containing genes presumed to be found in all isolates of a species, often called "housekeeping" genes (Martin et al., 1998).

With whole genome sequencing becoming more common sequencing data for the entire genomes for the bacterial isolates are becoming available. This makes it possible to divide bacterial isolates into subtypes by defining MLST schemas based on more than 7 loci. Better resolution for the allelic profiles can be obtained and the discriminatory power for the method increases with more loci in the schema (Maiden et al., 2013). Whole genome sequencing made core genome MLST (cgMLST) and whole genome MLST (wgMLST) possible. Schemas for cgMLST are made by using loci for genes with high conservation, found in most of the genomes in a set of isolates from the species (Maiden et al., 2013). For example, for *L. monocytogenes* a core genome has been defined as genes found in over 95% of a set of 957 genomes, and a cgMLST schema of 1748 loci with accompanying core genome sequence types has been defined by the Institute Pasteur (Moura et al., 2016). MLST methods using schemas containing loci for all genes found in a set of isolates for a species are referred to as a wgMLST, and schemas like this can increase the discriminatory power of the method even more (Maiden et al., 2013).

1.3 Source attribution

The purpose of source attribution methods is to give information on the sources of foodborne pathogens infecting humans (Pires et al., 2009). The information gain from source attribution depends on the chosen method, the data availability, and the subject of study (Pires et al., 2009). The transmission chain of infection is the route between sources where the bacteria might have been introduced and the infected humans, and the sources defined in the source attribution method can be from any place along this transmission chain (Pires et al., 2009). An example of a transmission chain is the connections between natural environments, farmland and animals, food production facilities, food products, and infected humans (Pires et al., 2009).

Microbial subtyping methods like genotyping can be used in source attribution when genomic data is available for the bacterial isolates from the sources. Source attribution methods using genomic data for microbial subtyping rely on a relationship between genotypes of the bacteria and specific sources, giving a lower diversity of genotypes within each source and higher between the sources (Pires et al., 2009). The methods require a large data set of bacteria with known sources to make predictive models, and the models can only calculate probabilities for the predefined sources (Pires et al., 2009).

The models that use microbial subtyping have been divided into two general types of methods and are reviewed with other methods in "Critical Orientation in the Jungle of Currently Available Methods and Types of Data for Source Attribution of Foodborne Diseases" by Mughini-Gras et al. (2019). The two main methods in the review article are frequency matching and population genetics, and they are described below:

The *frequency matching methods* uses the frequency of distinct genotypes for its calculation of probable sources for clinical cases. The genotype frequencies per source are compared to genotype frequencies in the clinical cases. By weighing the frequencies by factors such as exposure and how common a source is the predictions can be more accurate. The frequency of clinical cases attributed to each source can then be calculated.

Population genetics methods consider each source as a population of isolates. The method uses distinct genetic markers like allele type for a certain locus as a trait. The trait needs to be

found in the genotypes of the isolates within each population and not found in the other populations. The importance of the trait depends on how many of the isolates within the source that have the trait. The probabilities of belonging to the different sources are calculated for the clinical cases based on which of the traits are found in their genotypes and the traits importance.

In newer approaches of source attribution association between genomic data for bacteria and sources are looked for using machine learning algorithms (Arning et al., 2021; Bayliss et al., 2023; Castelli et al., 2023; Lupolova et al., 2019; Munck et al., 2020; Tanui et al., 2022).

1.4 Machine learning

Machine learning methods are mainly divided into two types: supervised and unsupervised (Lupolova et al., 2019). Supervised methods can be used for prediction, and unsupervised methods can be used to look for grouping in the data (Lupolova et al., 2019). The samples that the methods use as input are described by their features, and the features are used as input variables for the methods (Lupolova et al., 2019). For supervised methods, where the goal is prediction, the outcome values for the predictions also need to be used as an output variable. The metadata associated with the input data usually contains this information (Lupolova et al., 2019). Depending on the metadata the predictions are either classification if the output values are categorical, or regression if the output values are continuous (Raschka & Mirjalili, 2019, pp. 1-7). When a model finds patterns in the feature variables depending on the output variable the method is supervised (Raschka & Mirjalili, 2019, pp. 1-7). In unsupervised methods output variables are not used to guide the model to find pattens, and unknown groups often referred to as clusters might be discovered in the input data (Raschka & Mirjalili, 2019, pp. 1-7).

1.4.1 Machine learning methods

1.4.1.1 Unsupervised machine learning

Unsupervised machine learning is useful for exploring the input data or when metadata for predictions is not available (Lupolova et al., 2019). It can be used to look for potential structuring in the data and give insight into the data set's potential to divide the data into

clusters (Lupolova et al., 2019). In unsupervised machine learning methods no output variables are given to the model, so the clusters are independent of any information in the metadata (Lupolova et al., 2019). Some common methods that can be used for these purposes are described below.

K-clustering methods are designed to be given a predefined number of k clusters to look for. A similarity measure is calculated between the input data and the chosen, often random at the beginning, centroids for the k clusters, then the samples in the input data are assigned to the cluster with the centroid they are most similar to (Raschka & Mirjalili, 2019, pp. 353-382). By updating the centroid values using the samples clustered to it the similarity measure between the samples and centroids can also be updated, and samples can be reassigned to the clusters again based on the new similarity measures. This can be repeated for a set number of iterations, or until the centroids no longer change when updated (Raschka & Mirjalili, 2019, pp. 353-382). The method for the centroid update, and the similarity measure depends on the chosen method.

In *hierarchical clustering methods* there are no need to predefine a set of clusters to look for (Lupolova et al., 2019). The most common way to use hierarchical clustering is agglomerative (Lupolova et al., 2019). In agglomerative hierarchical clustering all samples in the input data represent unique clusters at the beginning, then a similarity measure is computed between all samples and the most similar samples are clustered together, this goes on by clustering similar clusters together in a hierarchical way until every sample is in one large cluster (Raschka & Mirjalili, 2019, pp. 353-382). In hierarchical clustering the similarity measure and how clusters are linked can vary. A common way to cluster similar clusters together is to calculate the average of the similarity measures between the samples in one cluster and the samples in another cluster (Raschka & Mirjalili, 2019, pp. 353-382). The less common divisive hierarchical clustering goes the other way, from one large cluster to more and more clusters until all samples are their own unique cluster (Raschka & Mirjalili, 2019, pp. 353-382).

1.4.1.2 Supervised machine learning

There are many different supervised machine learning methods to choose from, and when choosing methods the data set and the purpose of the analysis should be considered (Lupolova et al., 2019). Supervised machine learning models that are used for classification are trained on input data with a known output class variable to find patterns that separate the data by class (Raschka & Mirjalili, 2019, pp. 1-7). The trained models can then be used for prediction of the class for new data samples. A few relevant methods for this study are presented below.

The Random Forest classifier is an ensemble learner method that combines the single prediction of a set of decision tree models to classify samples (Raschka & Mirjalili, 2019, pp. 91-103). The single decision trees are only trained on a subset of the original training samples, this makes the predictions based on different grounds for the individual trees (Raschka & Mirjalili, 2019, pp. 91-103). An illustration of a Random Forest model with 3 decision trees is shown in Figure 1, note that a Random Forest model can be an ensemble of several hundred decision trees. The decision trees are made of a binary tree structure of nodes, splitting the training samples in two at each node according to a set condition. Each condition is trying to find patterns in a random selection of the features that splits the samples by their class, in the final nodes the training data is ideally separated according to class (Raschka & Mirjalili, 2019, pp. 91-103). A function is used to ensure that the condition leaves the nodes after the split with higher purity of classes than before. These functions are called impurity functions and are used to calculate the purity of a node by measuring how many samples of different classes are left in each node, and an impurity of 0 means the node only contains samples from one class (Raschka & Mirjalili, 2019, pp. 91-103). When the Random Forest model is predicting the class for a sample the ensemble of decision trees might have some differences in prediction, but the final prediction of the class is settled by choosing the class that most trees predict. Based on the distribution of predictions among the decision trees the Random Forest model also comes with a prediction likelihood for all classes, not only a prediction for most likely class (Raschka & Mirjalili, 2019, pp. 223-237).

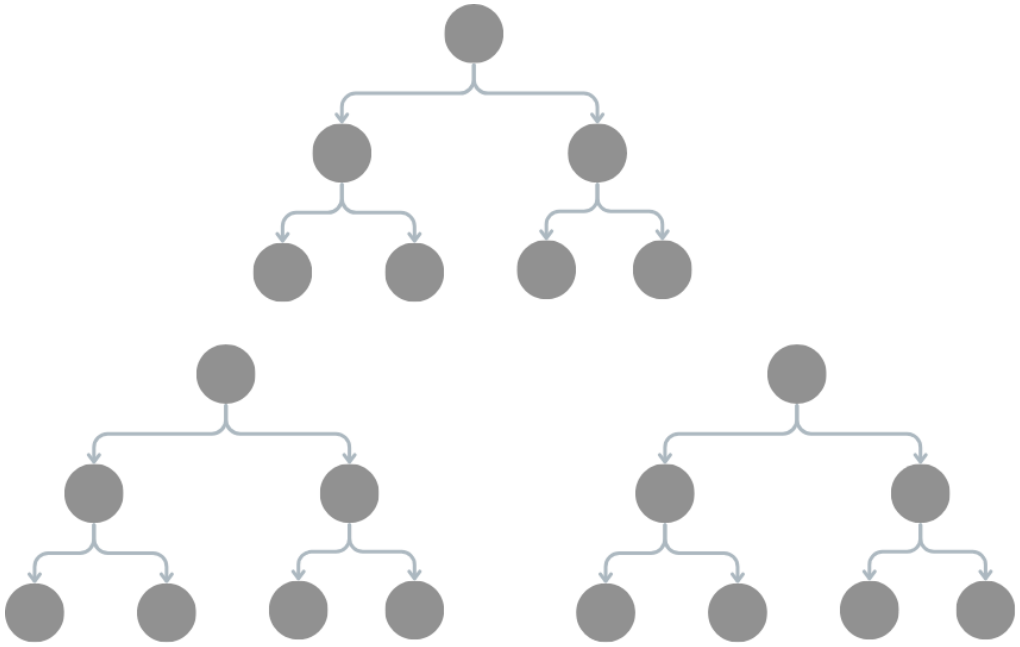

Figure 1. Illustration of a Random Forest model with 3 decision trees, the nodes in each tree sets a condition trying to separate the samples by class aiming to have a pure class in the last nodes. The dept of the tree can vary and depend on the number of nodes needed to set enough conditions to separate samples by class. The Random Forest model predicts the sample to be of the class the majority of the decision trees predict it to be. Created with canva.com/draw/.

The "gini" impurity is a common choice for the impurity function for the decision trees, and it tries to minimize the probability of misclassification (Raschka & Mirjalili, 2019, pp. 91-103).

The "gini" impurity is calculated by the following formula:

$$I_G(t) = \sum_{i=1}^{c} p(i|t)\,(1 - p(i|t)) \qquad (1)$$

Where I_G is the impurity in node *t*, *c* is all the classes and *p* is the proportion of class i in node t (Raschka & Mirjalili, 2019, p. 92).

Along with the impurity function the number of decision trees are common hyperparameters to tune to optimise the Random Forest model during training. A larger set of decision trees often gets better predictions but it comes with a higher use of computational resources (Raschka & Mirjalili, 2019, pp. 91-103).

The Support Vector Machine classifier finds decision boundaries in the feature space of the training samples to separate the samples by class (Raschka & Mirjalili, 2019, pp. 79-90). To find the optimal decision boundary between the samples the model tries to find the boundary that maximises the space between it and the samples closest to it in the feature space. The space between the boundary and the closest samples is called a margin, and the closest samples are called the support vectors (Raschka & Mirjalili, 2019, pp. 79-90). In Figure 2 a visualisation is presented of samples from two classes, circles and triangles, separated by a decision boundary in a dashed line, and with the maximised margin shown as solid lines.

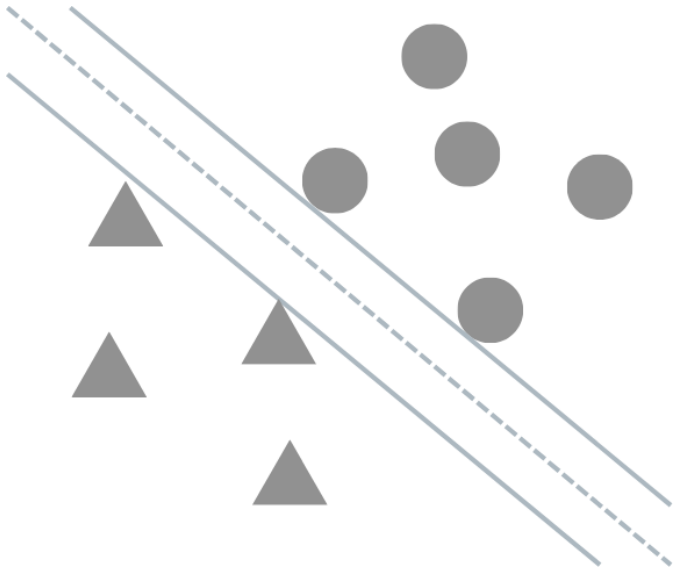

Figure 2. Illustration of how Support Vector Machine models separates two classes, here represented by circles and triangles, using a decision boundary shown by the dashed line found by maximising the margin between the classes shown as the space between the solid lines. Created with canva.com/draw/.

In Figure 2 the decision boundary is a line in a two-dimensional feature space separating only two classes, in a multi-dimensional feature space the boundary would be a hyperplane in the space, and by using a one-versus-rest approach several boundaries can be found if the training data has more than two classes. A one-versus-rest approach picks one class at the time to be the positive class and the rest of the classes are the negative class. This is then used to train a model and find a boundary to separate the positive class from the rest, and when all classes have been used as the positive class several decision boundaries have been found (Raschka & Mirjalili, 2019, p. 31).

Support Vector Machine models can use non-linear kernel functions to find non-linear decision boundaries (Raschka & Mirjalili, 2019, pp. 79-90). A kernel function is a similarity function which calculates similarity between features to make an abstract higher-dimensional space, and in this space the decision boundaries can be established (Raschka & Mirjalili, 2019, pp. 79-90). The decision boundaries can then be used to predict the class for new samples.

A common choice of non-linear kernel function is the "radial basis function" (rbf) kernel, and the similarity between the features is calculated by the following formula:

$$K\left(x^{(i)}, x^{(j)}\right) = \exp\left(-\Upsilon\left|\left|x^{(i)} - x^{(j)}\right|\right|^2\right) \quad (2)$$

Where $x^{(i)}$ and $x^{(j)}$ are different features and Υ (gamma) is a scaling hyperparameter for the rbf-kernel (Raschka & Mirjalili, 2019, p. 87). The similarities calculated between features using this kernel function makes the similarity between 0 and 1, where 1 is between identical features (Raschka & Mirjalili, 2019, pp. 79-90). The scaling hyperparameter is usually tuned to optimise the Support Vector Machine model during training and is a weight for how important the samples are. A high gamma gives tighter boundaries around the training samples, and a small gamma gives a wider boundary and therefore a softer separation of the samples (Raschka & Mirjalili, 2019, pp. 79-90).

Neural networks consist of layers of nodes where all the layers are connected. Their combination and interaction make it possible for the model to find non-linear patterns to separate the training samples by the classes in the output variable (Raschka & Mirjalili, 2019, pp. 383-423). Only the general terms of neural networks are explained here as numerous different variations of neural networks can be made depending on the architecture of the layers. Neural networks are generally built up with an input layer, at least one hidden layer, and an output layer (Raschka & Mirjalili, 2019, pp. 383-423). An illustration of the architecture of a shallow neural network with one hidden layer is presented in Figure 3.

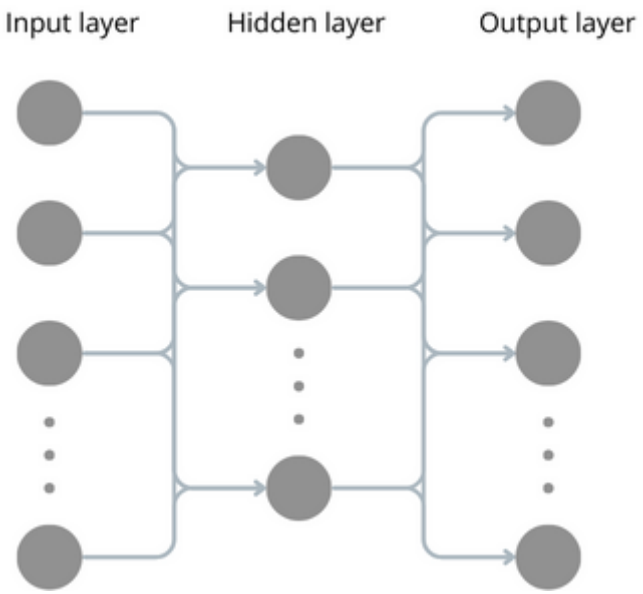

Figure 3. Illustration of the architecture of a neural network with input layer, one hidden layer and output layer. The arrows illustrate the interactions between layers when input from the samples goes through the network. The number of nodes in the layers can vary. Created with canva.com/draw/.

A neural network is trained by running the training data through the model for several iterations and letting the model find patterns in the data. During the training a loss function is used to calculate model performance often called loss, and the calculated loss is used by an optimiser to lead the learning to better performance (Raschka & Mirjalili, 2019, pp. 383-423). Each node in a hidden layer combines the features of a sample by weighing them and sends the combination to an activation function. Activation functions are used in the hidden layers to determine the output of each node, and in the output layer to determine the prediction (Raschka & Mirjalili, 2019, pp. 383-423). There are a number of adjustments that can be done for the neural network like changing the number of hidden layers and nodes, different kinds of layers and activation functions can be chosen, the batch size for how many samples are run through the model before the loss is calculated, and the learning rate for the optimiser can be tuned to get better predictions depending on the data and the purpose of the prediction (Raschka & Mirjalili, 2019, pp. 383-423).

The "Rectified Linear Unit" (ReLU) function is often used as the activation function after hidden layers, it is a non-linear function that aids the neural network to find non-linear patterns for classification (Raschka & Mirjalili, 2019, pp. 468-469). ReLU-functions makes

the output of the node 0 if the input to the function is lower than 0, and the output of the node is not changed by the function if the input to the function is equal to or greater than 0 (Raschka & Mirjalili, 2019, pp. 468-469). When using neural networks for classification with more than two classes the activation function after the output layer is usually "softmax". It outputs the probability of each class (Raschka & Mirjalili, 2019, pp. 465-466).

For classification with more than two classes the loss function "categorical cross-entropy" is mostly used, often with the optimiser "Adam" (Raschka & Mirjalili, 2019, pp. 486-489, 539-541). This loss function calculates the difference between the predicted classes and the true classes for the model during training. The optimiser uses this information to lead the training to better predictions (Raschka & Mirjalili, 2019, pp. 383-423).

1.4.2 Overfitting and underfitting

When supervised machine learning models are trained overfitting can become a problem. This means that a model finds too complex patterns specific to the data it is trained on (Raschka & Mirjalili, 2019, pp. 75-76, 127). An overfitted model would generalise poorly because it classifies by patterns not general to the classification task it is made for, but specific to one data set (Lupolova et al., 2019). The risk of overfitting can be minimised by using different strategies specific to each machine learning method. A lot of methods have a hyperparameter that can be tuned during training to avoid overfitting. Overfitting is rarely a problem for Random Forest models, the combination of many decision trees makes the models resistant against overfitting because noise influencing the single decision trees will not influence the models unless a majority of the trees are influenced (Raschka & Mirjalili, 2019, pp. 91-103). Support Vector Machine models have a cost hyperparameter regulating the cost for misclassification to allow for softer boundaries in the models (Raschka & Mirjalili, 2019, pp. 79-90). This can prevent the models from learning specific patterns for the data they are trained on by allowing some misclassification. In neural networks special dropout layers can be added. Dropout layers drops the output for a fraction of the nodes. The dropped nodes are chosen randomly and changes during the training (Raschka & Mirjalili, 2019, pp. 536-539). The models then have to rely on a changing set of nodes making it more general and robust against overfitting. Underfitting is the opposite of overfitting and occurs when too little information is held by the features or there are too few features to find patterns, making the

models too general to distinguish between classes (Raschka & Mirjalili, 2019, pp. 75-76). Underfitting can be prevented by choosing features with a high content of information, or adding more features if the models are too simple.

1.4.3 Preprocessing for machine learning

Most machine learning methods requires the input data to be presented as numbers (Raschka & Mirjalili, 2019, pp. 115-121), but sometimes the samples features are represented by categories of not numeric data. Categorical data are divided into ordinal and nominal data. In ordinal data there is a rank order between the categories, and in nominal data there is no order to the data (Raschka & Mirjalili, 2019, pp. 115-121). Ordinal data can be academic degrees like a bachelor's, a master's or doctorial. It could make sense to rank them and label them by numbers like 1, 2, 3 because a doctoral labelled 3 is a higher degree than master's labelled 2, and a master's is higher than a bachelor's labelled 1. If the data presents bacteria species like *Listeria monocytogenes*, *Salmonella enterica* and *Campylobacter jejuni* it would not make sense to rank them.

When categorical features are used in machine learning they often need to be encoded by transforming the categories into numbers. There are different ways of transforming the categories in categorical features to numbers by using encoding methods. Ordinal encoding simply gives a distinct integer to each category in a feature, and is suitable when the data is ordinal (Raschka & Mirjalili, 2019, pp. 115-121). Using an ordinal encoding for nominal data can make some machine learning methods misinterpret the relationship between the samples by assuming a rank order between the categories of the feature (Raschka & Mirjalili, 2019, pp. 115-121). One-hot encoding can instead be used to encode the features so that the categories will be represented as vectors. One-hot encoding creates vectors with length equal to the number of unique categories in the feature it encodes. The vectors are filled with zeroes and a single one, and the position of the number one holds the information on which category it is (Raschka & Mirjalili, 2019, pp. 115-121). For example, category 1 of 5 would be encoded as [1 0 0 0 0], and category 2 of 5 would be encoded as [0 1 0 0 0] and so on making one unique vector for each category.

Some machine learning methods are also sensitive to the scale of the features (Raschka & Mirjalili, 2019, pp. 124-127). Methods for scaling can be used to bring features into the same range. Standardisation of the features is common to use for scaling.

Standardisation can be done by the following formula:

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x} \tag{3}$$

Where x is the feature's value for sample i, and μ_x is the average value for the entire feature, and σ_x is the standard deviation (Raschka & Mirjalili, 2019, p. 126). Standardisation alters the values to centre the feature to have an average value of 0 and a standard deviation of 1.

1.4.4 Dealing with high dimensionality data

When the number of features are high compared to the number of samples this is often referred to as the "curse of dimensionality" (Raschka & Mirjalili, 2019, p. 107). High dimensionality makes it harder for the machine learning models to generalise, and often makes the performance of the models poorer. Models trained on high dimensional data have an increased risk of overfitting to the data set (Lupolova et al., 2019). Many features in a data set also makes maintaining and use of the data harder by increasing the need for computational time and resources (Raschka & Mirjalili, 2019, pp. 127-146).

With the purpose of reducing the number of features in a data set feature-selection methods can be implemented (Raschka & Mirjalili, 2019, pp. 127-146). There are several methods for selecting features, and the choice of method depend on the data. Getting an insight into the information held by the features of the data set is important to select meaningful features. The amount of information held by each feature can be explored by calculating the variety of values in a feature using diversity measures, like Shannon entropy, for the features (Lupolova et al., 2019).

Shannon entropy for a categorical variable can be calculated by the formula:

$$H(X) = \sum_{i=1}^{S} (-P(i) \log(P(i))) \tag{4}$$

Where *H* is the entropy of variable *X*, *S* is the set of distinct values in variable *X*, and *P* is the probability of value *i* (Chambert-Loir, 2022, pp. 23-54). For a variable an entropy of 0 would mean no variation among the samples (Chambert-Loir, 2022, pp. 23-54). Features with low diversity can be filtered out by using tools to measure and set a threshold for too low variation.

Mutual information can be calculated to not only take the total diversity of a feature into account, but also the feature's ability to give information on the output variable. Mutual information is a measurement of dependency between two variables, and gives a score for the shared information between two variables (Sherwin, 2010).

Mutual information between two variables can be calculated by the formula:

$$MI = H(X) + H(Y) - H(X, Y) \quad (5)$$

Where *MI* is the mutual information and *H(X)* is the entropy of variable *X*, and *H(Y)* is the entropy of variable *Y*, and *H(X,Y)* is the entropy of the pair (*X* ,*Y*) (Chambert-Loir, 2022, pp. 23-54). The entropy of the pair (*X, Y*) is derived from the *H(Y)* and the *H(X |Y)* which is the entropy of variable *X* given variable *Y* (Chambert-Loir, 2022, pp. 23-54). If there are no dependencies between variable *X* and variable *Y*, the entropy of the pair (*X* ,*Y*) will be equal to the two individual entropies added together, and the equation (5) will make the mutual information score 0. If knowing variable *Y* gives lower entropy for variable *X* there is a dependency between the variables making the entropy of the pair (*X* ,*Y*) lower than the two single entropies added together, and the mutual information is calculated to be higher than 0.

1.4.5 Assessing model performance

When training supervised machine learning models the hyperparameters of the models need to be optimised to get the best performing models (Lupolova et al., 2019). Hyperparameters can be the number of decision trees for Random Forest models, misclassification cost and scaling of the kernel function for Support Vector Machines, or optimiser learning rate and architectural choices like number of nodes in the layers for a neural network. In k-fold cross-validation the samples in the training data are divided into groups called folds, and the data is

normally split into 5-10 folds (Raschka & Mirjalili, 2019, pp. 191-211). All but one fold are used in the training and the last fold is used to test the predictive ability of the model using a performance metric. This is an iterative prosses which is usually repeated until all folds have been the test-fold, and the average performance of the iterations is calculated (Raschka & Mirjalili, 2019, pp. 191-211). The model hyperparameter can then be changed and the k-fold cross-validation can be run again. By comparing the performance for each cross validation run the hyperparameter setting providing the highest performance score can be chosen for the model. The possible combination of different hyperparameters that leads to the highest performance can be assessed by using a grid search. A cartesian grid search tries all combinations of hyperparameters when training the model, and in combination with the k-fold cross validation the grid search can evaluate and choose the ones that give the highest performance (Raschka & Mirjalili, 2019, pp. 191-211).

When validating the optimised model's predicting ability, it is good practise to do so with data that was not included in the training of the model (Raschka & Mirjalili, 2019, pp. 191-211). The aim is to avoid bias in the model and to see how well the model generalise (Raschka & Mirjalili, 2019, pp. 191-211). This can be achieved by splitting the data samples into a training set and a test set before the training, usually with the training set containing around 70% of the total data set's samples (Raschka & Mirjalili, 2019, pp. 121-124). When the model predicts the class for the samples in the test set, performance metrics can be calculated to analyse the model's strength.

Several statistical metrics are available for the assessment of model performance. Accuracy is perhaps the most intuitive measure as it is the fraction of correct predictions out of all predictions (Raschka & Mirjalili, 2019, pp. 211-222). Accuracy might not be the best measure to assess the performance when the classes that are predicted are not represented by equal number of samples in the data set. The model tends to be biased towards classifying the largest classes to get a high accuracy (Raschka & Mirjalili, 2019, pp. 211-222). Metrics as precision and recall can be used to get a better prediction on imbalanced data sets. These metrics use true and false positives and negatives for calculation (Raschka & Mirjalili, 2019, pp. 211-222). A positive prediction for a class means the model has predicted this class for a sample. If it is a true positive the prediction was correct, if incorrect it is a false positive. The

same principle applies to negative predictions for a class. The connections are presented in Table 2.

Table 2. Relationship between prediction and real value for classes in a data set.

	Positive prediction	**Negative prediction**
Real positive	True Positive (TP)	False negative (FN)
Real negative	False positive (FP)	True negative (TN)

Precision is calculated with the formula:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

Precision is the fraction of true positives for a class out of all the positive predictions of that class, meaning out of both true and false positive predictions (Raschka & Mirjalili, 2019, pp. 211-222).

Recall is calculated with the formula:

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

Recall is the fraction of true positives for a class out of all real positive for that class, meaning out of both true positives and false negatives (Raschka & Mirjalili, 2019, pp. 211-222).

Precision is affected by the model's false positive rate, and the recall is affected by the false negative rate. The two metrics often need to be balanced because wanting a high precision often comes with the price of a lower recall and vice versa. The F1-score is a performance

metric that does this by balancing precision and recall by combining them (Raschka & Mirjalili, 2019, pp. 211-222).

The F1-score is calculated with the formula:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (8)$$

The precision and recall, and therefore F1-score are calculated per class. The overall performance for a model with several classes can be found by using the averages of this metrics. In a macro average the individual metrics, like the F1-score for each class, are added and then divided by the number of classes (Raschka & Mirjalili, 2019, pp. 211-222). A weighted average is when the individual metrics first are weighted by the number of sample per class, before the metrics are added up (Raschka & Mirjalili, 2019, pp. 211-222).

1.5 Machine learning for source attribution

A supervised machine learning model for source attribution can be trained with genomic data from *Listeria monocytogenes* isolates with known sources to find patterns in the genomes that can classify the isolates by source. When a new clinical case of listerioses occurs, an isolate can be sampled from it and run through the model. Depending on the genome of the new isolate its source will be predicted by the model based on its training. When making a model like this several elements must be considered to make good choices for the model, and certain requirements for the data should be met to be able to fit it to a machine learning model and to use it for source attribution. This section will reflect upon these elements and requirements based on previous research in the field.

First, there has to be some genetic diversity in the data set for it to be possible for the machine learning model to look for general patterns in the genomes associated with the sources. The information held by the features are used by the model to find patterns, and a model with too few informative features is likely to be underfit (Raschka & Mirjalili, 2019, pp. 75-76). *L. monocytogenes* is a bacterium found in very different environments, this is an indication that the bacterium has adapted to a range of different habitats (Tortora et al., 2004, pp. 210-243).

For studies using MLST methods with only a few loci for defining the allelic profiles and sequence types (STs), the STs are often clustered into clonal complexes (CCs) based on similarity to further compare groupings of the isolates. Studies where frequencies of CCs and sources for *L. monocytogenes* are compared like the study by Linke et al. (2014), Maury et al. (2016) and Painset et al. (2019) suggest that specific CCs are found in higher abundance among the isolates in certain sources like natural environments, food processing environments and clinical human cases. This indicates that there is some structuring of isolates based on sources already at the resolution level provided by allelic profiles using MLST methods with only a few loci. Further, in a study by Moura et al. (2016) cgMLST was used to make allelic profiles for *L. monocytogenes* isolates and showed the usefulness of cgMLST to express the genetic diversity in a large set of isolates from several sources including clinical human cases, food products and processing environments, and animals. The study compared the discriminative powers between cgMLST and PFGE showing the advantages of cgMLST for surveillance of *L. monocytogenes* because of its ability to make profiles for the isolates with a higher resolution. Diversity in the bacteria population was also found in a study into genomic variation in *Listeria* species by Liao et al. (2023) where *L. monocytogenes* isolates from natural environmental sources and food-associated sources were compared. Several analyses into the core genes of the isolates, among them differences in cgMLST profiles, showed a difference between isolates from the two sources indicating core genes had an association with source. The study further investigated the association of accessory genes, genes that are not shared by a high fraction of isolates like core genes, and the source the isolates came from. They also found accessory genes associated with source for *L. monocytogenes.* MLST methods have shown the potential to express genetic diversity and information in genomic data from *L. monocytogenes*, and making allelic profiles also translates the genomic sequences into data suitable for machine learning methods.

The MLST methods of representing genomes as allelic profiles transforms the sequences into numbers, and cgMLST has been used several times before as input data for source attribution using machine learning. Examples of this are the use by Tanui et al. (2022) for source attribution of *Listeria monocytogenes*, and the use by Munck et al. (2020) for source attribution of *Salmonella typhimurium*. Several methods of representing the genomes, and among them cgMLST profiles, were compared by Arning et al. (2021) for source attribution of *Campylobacter jejuni*. In the study by Arning et al. (2021) the highest performance in the

study was found by using the cgMLST profiles directly. Using one of the MLST methods allelic profiles directly as input data like these studies would give categorical data for the features. The features would be the loci and each distinct number in a feature would represent one category of the alleles found at that locus. When having categories of allele types, it would not make sense to try and rank them in any order, so MLST methods would give nominal, categorical input variables for the machine learning model. It is important to be aware of the type of features that the MLST methods produce because many machine learning models would handle categorical features presented by numbers as ordinal (Raschka & Mirjalili, 2019, pp. 115-121). The way Random Forest models uses conditions to classify samples makes it a good fit for categorical features because no ordinal relationship is inferred even if the categorical features are presented as integer values (Wright & König, 2019). When using a Support Vector Machine model, the decision boundaries are found in the feature space of the samples (Raschka & Mirjalili, 2019, pp. 79-90). The model would therefore give ordinal meaning to categorical features presented as numbers by interpreting the relationship between the positions of the samples in the feature space. For this reason it is common to scale the input data for Support Vector Machine models to optimise the performance (Raschka & Mirjalili, 2019, pp. 124-126). The same applies to neural networks which uses the samples combination of features for training and therefore would use the feature values as numeric for categorical features presented as numbers (Raschka & Mirjalili, 2019, pp. 383-423).

Previous research using supervised machine learning for source attribution has shown that ensemble learners has performed well as models for source attribution. When using the cgMLST profiles of *L. monocytogenes* isolates as input data for machine learning models Tanui et al. (2022) compared the performance of three different ensemble learners, among them Random Forest, with a Support Vector Machine using a rbf-kernel. The result of comparing the models showed a higher performance for the ensemble learners. This was also the case for Arning et al. (2021) which compared 14 different machine learning methods for source attribution of *Campylobacter jejuni*, where the ensemble learner XGBoost performed best closely followed by Random Forest. The performance of the Support Vector Machine model using cgMLST profiles in the study by Tanui et al. (2022) had a little lower performance than the ensemble learners when attributing sources for *L. monocytogenes*, while the Support Vector Machine with a rbf-kernel in the study comparing 14 different machine

learning methods by Arning et al. (2021) was among the methods with lowest performance when attributing sources for *Campylobacter jejuni*. Support Vector Machine models with other representations of the genomes than MLST allelic profiles used as input data have also shown potential when used for source attribution. Predicted protein variants in *Salmonella enterica* isolates were used by Lupolova et al. (2019) to compare the source attribution performance of the methods Support Vector Machine, Random Forest and a neural network. The result of this study was a similar performance for all the methods involved. Neural networks for source attribution were also explored by Arning et al. (2021), and the study included 4 different neural network architectures which all performed quite similar. Their performances were a little lower than the ensemble learners when cgMLST profiles were used as input data.

Further, a method for feature-selection might be useful because most ways of representing genomic data make the data complex with a high dimensionality (Lupolova et al., 2019). While working with genomic data for one species it can be assumed that a lot of the features will contain similar values for all isolates, these features contribute with little information on dividing the isolates (Lupolova et al., 2019). In the source attribution study for *Salmonella enterica* by Lupolova et al. (2019) Shannon entropy was used to explore the diversity in the data set. To select features with high content of information both Munck et al. (2020) and Tanui et al. (2022) reduced the number of features in their cgMLST data sets by filtering out low diversity features, based on distinct values in each feature compared to number of isolates, and frequency ratios between the distinct values.

1.6 Aim of study

The main aim of this study was to explore supervised machine learning models to see if different methods were able to classify Norwegian *Listeria monocytogenes* isolates by source using whole genome sequencing data. A secondary objective was to investigate the diversity and the information held by the data set, because some diversity is required to separate the isolates by source.

The first step to achieve the aims were to make allelic profiles with both cgMLST and wgMLST for the genomes in the data set. The diversity and information captured by using cgMLST and wgMLST were compared by calculating diversity measures, and unsupervised machine learning methods were used to look for similarities and grouping of the isolates.

Further, the different machine learning methods Random Forest, Support Vector Machine and a neural network were used to make models and their performance were compared. When comparing the supervised machine learning methods in this study the Random Forest method was used as a benchmark to compare the other methods against, based on the experience obtained from earlier studies where tree-based ensemble methods for categorical data and with the direct use of allelic profiles as input data had been shown to work well (Tanui et al., 2022; Wright & König, 2019). Along with the different methods ability to handle the different allelic profiles directly, the methods performance on reduced sets of loci were compared by selecting subsets of the loci in the allelic profiles. The feature-selection was used to try and enhance the performance of the models as well as to explore if there was a threshold for how many loci needed to be present before the information in the data set got too low and affected the performance of the models.

The final step for obtaining the aims of this study were using the models to predict the sources from a set of isolates from clinical human cases of listeriosis.

2.0 Materials and methods

2.1 Software

The software used in this study and the most used libraries during coding are stated here. The software ALPPACA (Kaspersen & Fiskebeck, 2022) version 23.04.1 and chewBBACA (Silva et al., 2018) version 3.1.0 were run by UNIX commands and were used for making the different MLST allelic profiles. ALPPACA and chewBBACA utilized the high-performance computing resources provided by Sigma2 – the National Infrastructure of High-Performance Computing and Data Storage in Norway. The time and resources needed to run ALPPACA and chewBBACA made it most efficient to use them with the help of Sigma2. The coding for data preprocessing, exploration and machine learning were done using Python version 3.9.18 in Spyder version 5.4.3 on a windows computer with the processor 11th Gen Intel(R) Core(TM) i7-1165G7 and 16.0 GB RAM. Except for the neural networks which were run using Python version 3.10.12 in Google Colaboratory (https://colab.research.google.com). Neural networks often use more computational resources for training than other machine learning methods, and resources are made available for this purpose by Google Colaboratory. In Python the libraries pandas version 2.1.4, NumPy version 1.26.2, and functools version 1.6.4 were frequently used to manage the data. To aide in the exploring of the data the library SciPy version 1.11.4 was used, and a k-mode clustering was done using the library kmodes version 0.12.2. Most of the machine learning were performed using the library Scikit-learn version 1.3.0, but the architecture in the neural networks were made using the keras module from the TensorFlow library version 2.15.0.

Most of the data visualisations of the results were made using R version 4.3.1 in RStudio version 2023.06.1+524, and the libraries dplyr version 1.1.3 and tidyverse version 2.0.0 were used in R to manage and visualise the results. The exceptions, visualised using Python, were the heatmaps using the library seaborn version 0.12.2 and the confusion matrices made for visualising the machine learning performances using Scikit-learn.

Parts of the code are presented as Figures during the descriptions of the methods. The full source code using Python is available at GitHub:
https://github.com/teanderse/Listeria_sourceattribution

2.2 Data set

A total number of 1097 *Listeria monocytogenes* isolates were included in this study. The data set, containing whole genome sequencing of the isolates and metadata about their sources, was released from the study "Whole-Genome Sequencing Analysis of *L. monocytogenes* from Rural, Urban, and Farm Environments in Norway. Genetic Diversity, Persistence, and Relation to Clinical and Food Isolates" by Fagerlund et al. (2022). The assemblies of the genomes made of the isolates sequencing data were acquired from previous work done by the Norwegian Veterinary Institute using the same data set. The data set had 130 isolates sampled from Norwegian clinical cases. Further, the data set included 456 isolates sampled from Norwegian meat processing factories that were from the food processing environment, raw food material and cooked food products. It included 299 isolates sampled from Norwegian salmon processing factories that came from food processing environments, raw food material and food products. From the outdoor environment there were 110 isolates from rural and urban locations all over Norway. The rural and urban environmental isolates were sampled from grassland and animal path, residential areas, agricultural fields, locations near food processing plants, beach or sandbank, and forest or mountain area. There were 78 isolates sampled from cattle feed, faeces, milk filters, and teat swabs on Norwegian dairy farms. The last source were slugs with 24 isolates sampled from the species *Arion vulgaris* found in gardens and farm environments in Norway. The number and percent of isolates per source is presented in Table 3.

The 5 sources meat processing factories, salmon processing factories, rural and urban environments, dairy farms, and slugs constitute the classes the machine learning models will classify the isolates by. The isolates from clinical cases of listeriosis were not a part of model training, but later used for the models to attribute to a source.

Table 3. The number and percent of isolates per source in the data set used for this study.

Source	Number of isolates	Percent of isolates
Meat processing factory	456	42%
Salmon processing factory	299	27%
Rural/urban environment	110	10%
Dairy farm	78	7%
Slugs	24	2%
Clinical cases	130	12%
All	**1097**	**100%**

2.3 Preprocessing and data cleaning

2.3.1 Allelic profiles

The isolates genomes were transformed to allelic profiles using two types of MLST methods. Both cgMLST and wgMLST profiles were made for the isolates using the software chewBBACA (Silva et al., 2018). The chewBBACA software can either make the schema by itself by finding the loci of the genes of interest and then define the nomenclature for the alleles found at these loci, or an external schema can be provided for the software to use. The software then uses the given schema for its allele calling to match the alleles found at the loci of interest in each isolate to the schema and give each isolate an allelic profile.

ChewBACCA has an integration with the nomenclature server Chewie-NS (Mamede et al., 2021) for easy downloading of external cgMLST schemas to use for the allelic profiling. For making internal schemas, the software uses Prodigal (Hyatt et al., 2010) to predict coding sequences (CDSs) in each genome provided for schema creation. All distinct CDSs found are further processed by translating them to protein sequences and then clustering them based on alignment score ratios using Basic Local Alignment Tool (BLAST) (Altschul et al., 1990). BLAST aligns the sequences against each other and gives the alignments a score based on how similar the sequences are. The BLAST scoring ratio (BSR) is a normalisation of the raw BLAST scores making them range from 0 to 1, where 1 is defined as a perfect match between the sequences (Rasko et al., 2005). CDSs are defined as alleles of the same loci and clustered when their BSR is equal to or greater than 0.6. Further, only the largest allele in each cluster is kept representing the distinct loci in the schema. Alleles can be added to each locus defined in

the new schema by using the same input genomes to perform an allele call with the software using the new schema. After the allele calling paralogous loci can be removed based on alleles that matches more than one locus, indicating a gene duplication.

For both the cgMLST and wgMLST the allele calling is performed based on CDSs found by Prodigal (Hyatt et al., 2010) in the input genomes aligned against the alleles in the schema using BLAST (Altschul et al., 1990). The CDSs in the input genomes are first matched at a DNA sequence level. If they are not matched to an allele from a locus in the schema, they are translated to protein sequences and matched again at a protein level. Still unmatched alleles are assessed if they are novel alleles that not yet has been assigned a number in the schema. Novel alleles are in addition to being assigned a distinct number marked with a flag. Some of the isolates will for different reasons not have an allele call to all loci defined in the schema. Instead of a number for the allelic profile this locus will get a flag depending on the reason for the missing allele. Missing allele calls to a locus happens if there are no CDS match for the locus, the locus is not found or are at the tip of the assembly of the genomes, the CDS is smaller or larger than the matched allele from the schema, or there are several matches. All flags and their meaning can be found in the chewBBACA documentation file (https://chewbbaca.readthedocs.io/en/latest/index.html).

2.3.1.1 cgMLST

To obtain cgMLST profiles for the genomes the software tool ALPPACA (Kaspersen & Fiskebeck, 2022) was used. ALPPACA is a nextflow pipeline software tool which contains a cgMLST pipeline using chewBBACA. This pipeline was used to run the commands necessary to make cgMLST profiles. For the cgMLST profiles an external schema was used. The input commands for ALPPACA are shown in Figure 4. FASTA files of the 1097 *Listeria monocytogenes* isolates genomes were provided as the "$SAMPLE" for the *--input*, and to make sure results are reproducible, a prodigal training file for *L. monocytogenes* as the "$PTF" was provided for the *--ptf* parameter. The download of an external schema was triggered by *--download_external*, and the schema to download was specified in the command line as *--species_value* "6" for *L. monocytogenes* and schema number "1" as *--id_value*, and was from the nomenclature server Chewie-NS (Mamede et al., 2021). The chosen schema contained 1748 loci for core genes and is the Institute Pasteur *Listeria monocytogenes*

cgMLST schema (Moura et al., 2016). The parameter --*output_schema* adds the downloaded schema to the output folder, --*skip_schema_eval* stops the default schema evaluation as this is not necessary for the external schema, and --*prepped_schema* states that the schema is in the format necessary for chewBBACA, all external schemas from Chewie-NS is in the format used by chewBBACA. The --*max_missing* parameter makes a separate filtered output file without isolates having over the threshold, here set to "12", missing alleles, but this file was not used. The --*skip_mlst* parameter stops the default making of regular 7 loci MLST allelic profiles. The software also outputs a dendrogram using the clustering method defined as "nj" (neighbour joining) in the –*clustering_method* but it was not used in this study. The name "alpaca_run" was given to the output folder parameter --*out_dir*, and the main result file in this folder with the allelic profiles for all the input isolates were used further in this study. The wiki page of ALPPACA has a detailed description of all parameters for the software (https://github.com/NorwegianVeterinaryInstitute/ALPPACA/wiki).

```
$NEXTFLOW run $MAIN -c $CONFIG -profile apptainer --track cgmlst --input $SAMPLE --ptf $PTF --download_external \
--output_schema --skip_schema_eval  --species_value "6" --id_value "1" --prepped_schema --max_missing 12 \
--skip_mlst --clustering_method nj  --out_dir alppaca_run
```

Figure 4. Command line for running ALPPACA to make cgMLST profiles, the isolates genomes were the --input $SAMPLE, and --ptf $PTF was a prodigal training file. The additional parameters used were --download_external to states that a downloaded schema was used, --output_schema added the used schema file to the output folder, --skip_schema_eval stopped the default schema evaluation, --species_value "6" was the number for Listeria monocytogenes as species and –id_value"1" the schema number 1 from Chewie-NS, --preped_schema states that the schema form Chewie-NS already was in the right format, --max_missing "12" filtered out isolates with over 12 missing alleles, --skip_MLST stopped the default 7 loci MLST profiles from being made, --clustering_method "nj" outputs a dendrogram using neighbour joining as clustering method, and finally the --out_dir "alpaca_run" was the name of the output directory.

2.3.1.2 wgMLST

For the wgMLST profiles an internal schema was created using the schema creation command described earlier for the chewBBACA software. FASTA files for the 1097 *Listeria monocytogenes* isolates genomes were used as input "$SAMPLE" for the -*i* parameter, as shown in Figure 5. The same prodigal training file "$PTF" used for the cgMLST profiles was also provided here for the --*ptf* parameter. The names "chewbbaca_shema_run" and "schema_listeria_wgMLST" were the name of the output folder -*o* and file name for the created schema --*n*, and the number "20" for the –*cpu* parameter indicates CPUs used for the task. After the schema creation the output file contained the schema with FASTA files for the

alleles representing the loci, and the prodigal training file was also stored along with the schema for reproducibility. In addition to the schema the output folder had files for invalid alleles and metadata for the coordinates of the predicted genes in each genome.

```
chewBBACA.py CreateSchema -i $SAMPLE -o chewbbaca_shema_run --n schema_listeria_wgMLST --ptf $PTF --cpu 20
```

Figure 5. Command line for creating schemas with chewBBACA, the isolates genomes were the --input $SAMPLE, and --ptf $PTF was a prodigal training file. The --o "chewBBACA_shema_run" was the name of the output folder, and –n "schema_listeria_wgMLST" was the name of the file for the created schema. The parameter --cpu "20" states that 20 CPUs were used for the task.

The created schema only stored the largest allele in the cluster of alleles for each locus, so alleles were added to the schema by running chewBACCA's allele calling command with the same 1097 *Listeria monocytogenes* isolates genomes. The allele calling command was run twice because the same command was also used to make the wgMLST profiles. The command line is visualised in Figure 6 where "$SAMPLE" given to the input parameter *-i* were the isolates genomes as FASTA files and "$SCHEMA" was the schema created in the previous step given as input to the *-g* parameter. The output folder *-o* was called "chewbbaca_call_run" and after the second run of the allele calling command it contained the wgMLST profiles and possible paralogous genes. The number of CPUs used for the task were 20 as indicated by the parameter *--cpu* "20".

```
chewBBACA.py AlleleCall -i $SAMPLE -g $SCHEMA -o chewbbaca_call_run --cpu 20
```

Figure 6. Command line for allele calling with chewBBACA, the isolates genomes were the -input $SAMPLE, and the wgMLST schema created earlier was the $SCHEMA given as input to the parameter -g. The parameter -o "chewBBACA_call_run" was the name of the output folder. The parameter –cpu"20" states that 20 CPUs were used for the task.

Paralogous genes found during allele calling were removed from the wgMLST profiles using the gene removing command. Figure 7 shows the command line for removing paralogous genes. The wgMLST profiles used as input *-i* were called "results_alleles.tsv", and the paralogous genes called "paralogous_count.tsv" for the *-g* parameter were found during the allele calling and were in the output file from the allele calling command "chewbbaca_call_run". The output file *-o* was the wgMLST profiles without the paralogous

genes called “results_alleles_NoParalogs.tsv”. This output file with all the isolates and the paralogous genes filtered out of their allelic profiles were used further in this study.

```
chewBBACA.py RemoveGenes -i chewbbaca_call_run/results_alleles.tsv -g chewbbaca_call_run/paralogous_counts.tsv /
-o chewbbaca_call_run/results_alleles_NoParalogs.tsv
```

Figure 7. Command line for removing paralogous genes with chewBBACA, the input -i is the wgMLST profiles and -g the paralogous genes found during the allele calling. The output file -o named “results_alleles_NoParalogs.tsv” was the allelic profiles for the isolates with the paralogous genes removed.

A schema evaluation was done for the created schema by assessing the output evaluation file of chewBBACA’s schema evaluator command, the command to create a schema evaluation is shown in Figure 8. The schema “schema_listeria_wgMLST” created by the schema creation command was the “$SCHEMA” used as input *-i*, and the output evaluation folder *-o* was called “chewbbaca_schemeval” and the parameter *–cpu* “20” indicated 20 CPUs were used for the task.

```
chewBBACA.py SchemaEvaluator -i $SCHEMA -o chewbbaca_schemaeval --cpu 20
```

Figure 8. Command line for evaluating schemas with chewBBACA, the internally made wgMLST schema was the input -i as $SCHEMA and the output folder -o for the evaluation was called “chewbbaca_schemaeval”. The parameter –cpu”20” states that 20 CPUs were used for the task.

2.3.2 Data cleaning

After the allelic profiles were made the output for the cgMLST and wgMLST were two data sets that contained columns of loci and rows of isolates represented by their allele type numbers in the columns. If for some reason no allele in an isolate was found for one of the loci in the schema the allele number at that locus would be replaced by a flag representing the reason why the allele was missing. The metadata for the isolates containing a column for the sources associated with each isolate was added to the data sets as a class variable column holding the labels for the sources. For simplicity, all flags marking missing alleles were replaced with the not-a-number value “NaN”, and flags marking new alleles were stripped of the flag leaving only the number. After this the isolates with clinical cases as source were filtered out and stored separately, as they were not used to create machine learning models.

Several steps were taken to prepare the two data sets for machine learning. The cleaning steps to preprocess missing values were inspired by the preprocessing done by Arning et al. (2021). For the cgMLST data set a threshold was set to maximum 10% missing values for the rows with isolates. This was done to filter out isolates with potential errors since the loci represent core genes that are expected to be in over 95% of isolates (Moura et al., 2016), errors can happen during processing steps like sequencing or assembly of the genomes for the isolates. This filtering did not filter out any isolates and to ensure the same number of isolates in the cgMLST data set and the wgMLST data set this filtering was not done to the wgMLST data set. Also, the wgMLST data set possibly contained several loci for accessory genes, that were not expected to be present in high fractions of the isolates.

Further, columns containing loci with 10% or more missing values were filtered out to not include loci that few isolates had alleles for. The loci that were filtered out were considered to contribute with little genetic insight for the isolates. Because genes being absent might contribute to genetic insight the remaining not-a-number values were kept and replaced with the number -1 to give them a distinct value. As a last filtering step, a threshold was set to minimum 15 isolates per source to filter out potential sources with a low number of isolates for each data set. This threshold was set to ensure that the machine learning algorithms had enough data to train on for each source, and was the same threshold set by Arning et al. (2021).

2.3.3 Encoding the output class variable

Most of the Scikit-learn supervised machine learning methods for classification requires the labels in the class variable to be encoded into numbers. The class variable holding the sources labels for the isolates was therefore encoded with the LabelEncoder() function from Scikit-learn's module preprocessing to get one distinct number value for each source starting at the number 0. The LabelEncoder() function is made for this purpose and does not assume any order among the classes even though they are represented by numbers. The Scikit-learn's classifiers that will use the labels to train models are also designed to not infer any order from the class variable used for the source labels.

The neural network method used was from the TensorFlow library. When classification problems are solved with neural networks the output layer commonly uses a “softmax” activation function. Therefore, neural network models for classification often require the labels in the class variable to be one-hot encoded into vectors like the output of the “softmax” activation function. The class variable holding the labels for the sources were one-hot encoded using Scikit-learns OneHotEncoder() function from the preprocessing module to obtain this for the neural networks.

2.4 Data exploration

Different techniques of data exploration were used to get a better understanding of the cgMLST and wgMLST data sets. To get insight into the differences between how the two ways of making schemas impact the number representation of the distinct allele types the highest number values assigned to alleles in allelic profiles for each schema were compared. Further, diversity was calculated for each locus to compare the two data sets, and groupings and similarities among the isolates within each data set were examined by using k-mode and agglomerative hierarchical clustering.

2.4.1 Differences in MLST methods and diversity of data

By using an external schema for cgMLST and making an internal schema for wgMLST the nomenclatures for the different alleles were not harmonized, and to get an insight to how different the two schemas numbered their distinct allele types the 5 highest number values given to alleles were found for both the cgMLST and the wgMLST data sets. For each locus the number of distinct alleles were counted, and the highest number counted was recorded for each data set. This was done because the different representations of the data could impact some of the machine learning methods.

Further, the diversities of the different loci were investigated for both the cgMLST and the wgMLST data sets. The Shannon entropies were calculated for each locus in the data set. The entropy() function from SciPy stats module was used to do the entropy calculations. The

diversities found in each data set were compared to get an insight into the information captured by the two different MLST methods.

2.4.2 Clustering

Both k-mode clustering and hierarchical clustering were performed on the cgMLST and the wgMLST data sets to look for clustering of the 5 sources or other similarities that grouped the isolates within the data sets.

For k-mode clustering the KModes() function from the `kmodes` library was used. The code for setting up the clustering function and using it on the data set is shown in Figure 9. The number of clusters to look for was set to "5", and the initialisation for the centroids was set to "random". The function was set to start over 10 times by default trying to find the best random start. The seed "3" was chosen to ensure reproducibility of the random initialisations of the centroids. K-mode clustering counts matching features between isolates and the centroids to assign isolates to clusters based on similarity, then the centroids are updated to represent the mode of the isolates assigned to it. The update of the centroids was done until the isolates no longer were assigned to new centroids. The output was a list of the clusters the isolates were assigned to.

```
# setting seed for reproducibility because of random centroid start
np.random.seed(3)
# doing k_mode clustering for 5 clusters
km = KModes(n_clusters=5, init='random', verbose=1)
clusters = km.fit_predict(MLST_data)
```

Figure 9. Kode for k-mode clustering on MLST data set, with 5 clusters to assign isolates to, and a random start for the centroids. The isolates were assigned based on count of matching features, and the centroids were updated using the mode values of the isolates assigned to it.

The hierarchical clustering was accomplished using the `SciPy` library. The code for the calculations and the clustering is shown in Figure 10. First, Hamming distances were calculated between all isolates using the function pdist() with the distance metric set to "hamming", which measured normalised Hamming distances by finding the proportions of differences between the features of two isolates and collecting all distances in a matrix. The distances were turned into regular Hamming distances by multiplying the normalised values

by the number of features associated with the data set. The distance matrix was then input for the linkage() function. The linkage functions method was set to “average”, and this decides how distances are calculated when the distances no longer are between single isolates but clusters. An average linkage means that the distance between clusters were calculated by taking the average distances of the single isolates within the clusters. This distance was used in the clustering to join clusters hierarchically. The hamming distances and linkage calculations were used as input for the clustermap() function from seaborn which made a heatmap of distances together with a dendrogram showing the linkage of the hierarchical clustering. The isolates were coloured by their source labels to see if the isolates from the same source clustered together.

```
# calculating hamming distances
# normalised hamming distances
distances = pdist(MLST_data, metric= "hamming")
dist_tbl = squareform(distances)
# not normalised hamming distances
wg_featureLength = 2496
cg_featureLength = 1734
nonNormalised_dist_tbl = dist_tbl.copy()
nonNormalised_dist_tbl = nonNormalised_dist_tbl*wg_featureLength
nonNormalised_distances = squareform(nonNormalised_dist_tbl)

# defining linkage for clustering
linkage = hc.linkage(nonNormalised_distances, method='average')

 # giving label a colour in the dendrogram
 colour = sns.color_palette("colorblind", 5)
 colour_map = dict(zip(labels.unique(), colour))
 row_colors = labels.map(colour_map).to_numpy()

 # hierarchical clustering using hamming distances visualised in a heatmap
 clusterplot = sns.clustermap(nonNormalised_dist_tbl, row_linkage=linkage, col_linkage=linkage, row_colors=row_colors,
               cmap="mako", cbar_kws={"orientation": "horizontal",'label':'Normalised hamming distance'},
               cbar_pos=(.25, 0, .7, .03))
```

Figure 10. Code for hierarchical clustering with heatmap for distances of MLST data set. Normalised Hamming distances were first calculated, the distances were converted into regular Hamming distances before hierarchical clustering with average linkage. Here the distances are calculated for the wgMLST data set.

2.5 Feature-selection

Before feature-selection the cgMLST and the wgMLST data sets were split 70/30 in training and test data sets using Scikit-learns model.selection module’s function train_test_split(). The

function shuffles and randomly splits the isolates into the training and test data sets. The source labels were taken into account by the train_test_split() function by setting the stratify parameter to use the class variable holding the source labels when splitting. This ensured both data sets had isolates from all the sources. The training and test data sets were fixed by setting a seed to "3" in the function to ensure there was no data leakage between the training and test data sets when calling the function more than one time. This is considered a good approach in machine learning to make sure the test data set is not part of any model building and training, so the test performance calculations stay as unbiased as possible (Raschka & Mirjalili, 2019, pp. 11-14).

Several approaches to feature-selection were considered to try reducing the number of loci as features without losing information they might give on the sources. The Shannon entropy, earlier used for diversity measures, gives insight to the variety of values and information within each feature, but high variety for a feature alone might not be a good enough criterion for feature-selection as it does not give any information on the dependencies between a feature and the sources. Mutual information, which is based on entropy, was found as a better choice for feature-selection. Mutual information calculates the dependencies between two variables, and is a measurement of reduction of uncertainty in one variable when the value of another variable is known (Sherwin, 2010). Meaning that if the allele type of the locus in question is known and it can give information on what the source might be there are a dependency between the locus used as a feature variable and the class variable holding the source labels. The more shared information there is between the variables, the higher the mutual information is calculated to be. Mutual information can therefore help pick out the loci with values having an association to the source.

2.5.1 Feature-selection using mutual information

Only the cgMLST data set was used to make smaller feature selected data sets. The cgMLST data set was effectively speaking a subset of the wgMLST data set. The wgMLST data set was therefore used further with all its features. The subsets of the cgMLST data set were used to get an insight into how the different machine learning methods handled different number of features, and to see if feature-selection could improve the model performance compared to the cgMLST data set with all features.

The Scikit-learn function mutual_info_classif() from the feature_selection module was used along with the SelectPercentile() function from the same module to calculate the mutual information and to select the features with highest mutual information scores with the class variable. The code with the use of the functions is presented in Figure 11. The mutual_info_classif() calculated the mutual information shared between each feature and the class variable holding the source labels of the isolates. In the mutual_inf_classif() function the discrete_features option was set to "True" to state that the input data was categorical data. Then the SelectPercentile() function, based on the mutual information scoring of the features, selected the chosen percent of the features with the highest scores. Different feature selected training data sets were made with the features equal to the 10, 20, 30, 40, and 50 percent of the features with highest mutual information score in the original cgMLST training data set. The SelectPercentile() function was also used to transform the test data set to have the same features as the feature selected training data set before they were used to train and test machine learning models.

```
# feature selection only done for cgMLST data

# feature selection based on mutual information
# percentile best features
percentile_threshold = 50  #(10, 20, 30, 40 or 50)
pBest= SelectPercentile(score_func=partial(mutual_info_classif, discrete_features=True),
                        percentile=percentile_threshold)

# finding and reducing training set to p-best features
cgMLST_train_pBestReduced = pBest.fit_transform(MLST_train, labels_train)
# reducing test set based on calculation for best features done on training set
cgMLST_test_pBestReduced = pBest.transform(MLST_test)
```

Figure 11. Feature-selection on cgMLST training data set. The pBest object used the Select_percentile() function to select the features with the highest score, the percent of features chosen was decided by the percentile threshold. The mutual_info_classif() function calculates the features mutual information with the class variable holding the source labels and gives each feature a score. In the Figure the percentile threshold is set to 50. The pBest was fitted to and transformed the input training data to a training data set with the selected features. The pBest also transformed the test data set to have the same features as the training data set.

For further exploration of the features, the 10 percent of the features with highest mutual information score in the cgMLST training data set, were compared to the 10 percent of the loci calculated earlier to have the highest Shannon entropy for the entire cgMLST data set, to see if the loci with highest entropy also was among the features with highest mutual information score.

2.6 Machine learning models for prediction

The three machine learning methods: Random Forest, Support Vector Machine and a neural network were chosen to make models. These methods have different approaches for handling categorical input data, and the number of features in the input data. The Random Forest method was expected to handle both the categorical data and the high numbers of features well and was used as a benchmark for this study. The wgMLST training data set, the cgMLST training data set with all features and the different feature-selected cgMLST training data sets were used as input to compare the predictive abilities the models gained from being trained on data of different number of features and allelic profiles made by different MLST methods. The predictive performance of a model was assessed by training it on the training data set and using the test data set to evaluate the model's predicting performance. The performances of models trained on the different training data sets and with different machine learning methods were then compared.

The sections below describe how the training and testing of models for the three different methods were carried out.

2.6.1 Random Forest

The RandomForestClassifier() function from Scikit-learns ensemble module was used to make the model. A seed was set to "2" for the function to make sure the training was reproducible. This was necessary to avoid getting different decision trees each time the model was run. Different seeds were tested during the training to see if the seed impacted the model training performance before settling for one.

The GridSearchCV() function from Scikit-learns model selection module was used when training the models to optimise the hyperparameter. The model's hyperparameter for number of decision trees was adjusted to find the optimal value. The grid search used 5-fold cross validation repeated 10 times to evaluate the training performance of all hyperparameter values tested. The set up for the grid search used with the Random Forest model and fitted with the training data set is shown in Figure 12.

In the grid search the hyperparameter for number of decision trees was tested for the values 300, 400, 500, 600, 700 and 800. The impurity measure was not tested for different types but set to "gini". The optimal number of decision trees for the model was the number giving the highest average weighted F1-score from the cross validation during training. The time used for fitting the model with the grid search was recorded.

```
# setup for random forest classifier with seed for reproducibility
RF_model = RandomForestClassifier(random_state=2)

# parameter for RF_model
param_grid_RF = [{'n_estimators': [300, 400, 500, 600, 700, 800], 'criterion': ['gini']}]

# 5-fold cross validation with 10 repeats
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=3)

# gridsearch for best parameter search
gs_RF = GridSearchCV(estimator=RF_model,
                     param_grid=param_grid_RF,
                     scoring=({'weighted_f1':'f1_weighted', 'macro_f1':'f1_macro', 'accurcacy':'accuracy'}),
                     cv=cv,
                     refit='weighted_f1',
                     return_train_score=True,
                     n_jobs=-1)


# fitting model to training data set and finding best hyperparameters with grid search
gs_model_RF = gs_RF.fit(MLST_train, labels_train)

# defining optimal model
clf_RF = gs_model_RF.best_estimator_
```

Figure 12. Setup for grid search to fit Random Forest model. The model and the hyperparameter grid for different number of decision trees were input for the grid search. During training the grid search ran a 5-fold cross validation repeated 10 times to get an average performance for each hyperparameter value tested. The hyperparameter value giving the highest weighted F1-score was chosen as optimal for the model.

The Random Forest model fitted with the optimal hyperparameter was then used to predict the sources in the test data set to evaluate the model. The F1-score for each source, weighted F1-score and the macro F1-score were used to evaluate the model's performance when predicting the sources in the test data set.

2.6.2 Support Vector Machine

The Support Vector Machine model was made with the SVC() function from the module svm in Scikit-learn. The model was set up for both scaled and unscaled input data. The model with scaling was a part of a pipe with the Scikit-learns StandardScaler() function from the preprocessing module. The pipe leads the input data through the StandardScaler() function first and the output of the scaling is then used as input for the model. The StandardScaler() function standardised the features by subtracting the average value and divide by the standard deviation to centre and scale each feature to have a standard deviation of 1 before being input for the model.

The same setup for optimising the hyperparameters for the model during training was done using the GridSearchCV() function from Scikit-learns model selection module with 5-fold cross validation repeated 10 times to evaluate the training performance of all hyperparameter values and combinations of them. The hyperparameters optimised were the cost of misclassification and the gamma for scaling of the kernel. The set up for the grid search used with the Support Vector Machine model and fitted to the training data set is shown in Figure 13.

In the grid search the cost hyperparameter C was tested for the values 1.5, 2.0, 3.0, 3.5, 4.0, 5.0, 5.5 and 6.0, and the gamma hyperparameter for scaling the kernel was tested for the values 0.0005, 0.001, 0.002, 0.003 and 0.005. The kernel was not tested for different types but set to "rbf". The optimal hyperparameter combination for the model was the one giving the highest average weighted F1-score from the cross validation during training. The time used for fitting the model to the training data set with the grid search was recorded.

```
# setup for support vector clasifier
SVMno_model = SVC()
# setup for support vector pipeline with scaling
SVM_pipe = make_pipeline(StandardScaler(),
                         SVC())

# parameter range for C (cost)
param_rangeC  = [1.5, 2.0, 3.0, 3.5, 4.0, 5.0, 5.5, 6.0]
# parameter range for gamma for scaling of the rbf-kernel
param_rangeG = [0.0005, 0.001, 0.002, 0.003, 0.005]

# parameters
# for SVM_model with no scaling
param_grid_SVMno = [{'C': param_rangeC, 'gamma': param_rangeG, 'kernel': ['rbf']}]
# for SVM_pipe with scaling
param_grid_SVM = [{'svc__C': param_rangeC, 'svc__gamma': param_rangeG, 'svc__kernel': ['rbf']}]

# 5-fold cross validation with 10 repeats
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=3)

# chosing model with/without scaling and setup for grid
model = SVM_pipe      # SVMno_model for no scaling or SVM_pipe for scaling
grid = param_grid_SVM # param_grid_SVMno for SVMno_model or param_grid_SVM for SVM_pipe

# gridsearch for best parameter search
gs_SVM = GridSearchCV(estimator=model,
                  param_grid=grid,
                  scoring=({'weighted_f1':'f1_weighted', 'macro_f1':'f1_macro', 'accurcacy':'accuracy'}),
                  cv=cv,
                  refit='weighted_f1',
                  return_train_score=True,
                  n_jobs=-1)


# fitting model to training data set and finding best hyperparameters with grid search
gs_model_SVM = gs_SVM.fit(MLST_train, labels_train)

# defining optimal model
clf_SVM = gs_model_SVM.best_estimator_
```

Figure 13. Setup for grid search to fit Support Vector Machine model. The model can be in a pipe for scaling of the input data or without scaling. In the Figure the chosen model is the one with the pipe for scaling. The two hyperparameter grids were for different hyperparameter values for the cost "C" and the kernel scaling "gamma", one for the model without scaling and one for the model in the pipe with scaling. Both grids were set up to test the same values for the hyperparameters. The chosen hyperparameter grid in the Figure is the one for the model with the pipe. The model and its hyperparameter grid were input for the grid search. During training the grid search ran a 5-fold cross validation repeated 10 times to get an average performance for each hyperparameter combination tested. The hyperparameter combination giving the highest weighted F1-score were chosen as optimal for the model.

The Support Vector Machine model fitted with the optimal hyperparameters was then used to predict the sources in the test data set. The predicting performance on the test data set was evaluated by calculating the F1-score for each source, and the weighted F1-score and the macro F1-score for the model.

2.6.3 Neural network

The neural network model was built using the library TensorFlow's module keras. The setup of the shallow dense neural network that was made is presented in Figure 14. The model is shallow because it only has one hidden layer, and dense because all nodes in the hidden layer are fully connected to the next layer. The input dimension was equal to the number of features in the training data set. The model had one dense hidden layer followed by a dropout layer. The dense hidden layer used "relu" activation function as done in previous studies in the same field by Arning et al. (2021). The output layer was a dense layer with 5 nodes, one for each class of sources, with "softmax" activation function. To improve model performance the optimiser "Adam" was used with the loss function "categorical cross-entropy" also similar to the use by Arning et al. (2021) for their neural networks. The batch size was set to be equal to the entire training data set, and the epochs, meaning the iterations for the training, were set to 200 with early stopping after 10 iterations with no improvement of the calculated loss. The number of iterations was chosen by splitting the training data set into 80% training and a 20% validation sets. The model where then trained 5 times on the 80% training data set to find the approximate number of epochs where the weighted F1 performance score on the validation data sets stopped increasing. By also setting an option for early stopping the risk of overfitting was minimised. Neural networks need to be compiled or in other words put together to define the model before use. The function KerasClassifier() from Scikit-learn's wrappers module was used to compile the model, this made it possible to compile the model seen in Figure 14 during a grid search. The model can then be compiled with different values for the hyperparameters.

```
# function for making shallow dense neural network models and finding best hyperparameters with gridsearch
def create_shallowDenseNN(input_dim, neurons, dropout_rate ):
  ShallowDense_model = tf.keras.Sequential()
  ShallowDense_model.add(tf.keras.layers.Dense(neurons, input_dim=input_dim, activation="relu"))
  ShallowDense_model.add(tf.keras.layers.Dropout(dropout_rate))
  ShallowDense_model.add(tf.keras.layers.Dense(5, activation="softmax"))

  return ShallowDense_model

# set up for shallow dense neural network model
# input dimentions for different feature selected data sets of cgMLST or wgMLST
wgMLST = 2496
all_ = 1734
p10 = 174
p20 = 347
p30 = 520
p40 = 694
p50 = 867

ShallowDense_model = KerasClassifier(model=create_shallowDenseNN, input_dim=wgMLST, loss="categorical_crossentropy",
                                     optimizer=tf.keras.optimizers.Adam,
                                     metrics=["accuracy", f1_weighted, f1_macro], epochs=200, batch_size=676,
                                     callbacks=tf.keras.callbacks.EarlyStopping(monitor='loss', patience=10))
```

Figure 14. Setup for the shallow dense neural network. The model had an input dimension equal to the number of features in the training data set, here set to the wgMLST data set, one hidden layer with "relu" activation function, one dropout layer, and an output layer with 5 nodes and "softmax" activation function. The model was compiled with the "Adam" optimiser using the "categorical cross-entropy" as loss function. The iterations were set to 200 with early stopping after 10 iterations without improvement in the loss. The entire training data set is used as batch size.

The GridSearchCV() function from Scikit-learn's model selection module, with 5-fold cross validation, was used to determine the optimal hyperparameter value for number of nodes in the hidden layer, the dropout rate for the dropout layer, and the learning rate for the optimiser. The setup for the grid search is shown in Figure 15. Wider grids for the hyperparameter optimalisation were tested for the grid search before settling on testing the number of nodes for the values 70, 75 and 80, the dropout rate for the values 0.2 and 0.3, and the learning rate for the values 0.001 and 0.0001. When testing wider grids high and low values for the hyperparameters were tested together for two at the hyperparameter at the time, while the third was one fixed value in the grid search until the values were narrowed down. The grid search was computationally expensive and time consuming so finding fewer values to adjust the hyperparameters, and only tuning 3 of the hyperparameters were decided as enough to not increase the amount of computational resources and time consumption too much. The grid search was run 5 times to get an average of the cross-validation's weighted F1-scores because of the stochasticity of neural networks. Several parameters in a neural network have a random

start value, and it was not possible to control this by setting a seed. The values will therefore change a little bit every time the neural network is compiled even though the settings and input are the same every time. The optimal values for the hyperparameter combination were defined as the one giving the highest average weighted F1-score calculated for the 5 runs of the grid search. The time used for fitting the model to the training data set with the grid search was recorded. The hyperparameter combination with optimal values was chosen for the final training of the model.

```
# tune number of nodes, dropout rate and learning rate
# hyperparameter for SDNN_model
learning_rate = [0.001, 0.0001]
dropout_rate = [0.2, 0.3]
nodes = [70, 75, 80]
param_grid_SDNN = [{'model__neurons':nodes, 'optimizer__learning_rate': learning_rate, 'model__dropout_rate':dropout_rate}]

# gridsearch for best parameter with 5-fold cross validation
gs_SDNN = GridSearchCV(estimator=ShallowDense_model,
                  param_grid=param_grid_SDNN,
                  scoring=({'weighted_f1':'f1_weighted', 'macro_f1':'f1_macro', 'accurcacy':'accuracy'}),
                  cv=5,
                  refit='weighted_f1',
                  return_train_score=True)


# fitting model to training data set and finding best hyperparameters with grid search
gs_model_SDNN = gs_SDNN.fit(MLST_train, labels_train)
```

Figure 15. Setup for the grid search to fit the shallow dense neural network model. The model and the hyperparameter grid for different values for the learning rate, nodes in the hidden layer and dropout rate are input for the grid search. During training the grid search ran a 5-fold cross validation to get an average performance for each hyperparameter value combination tested. The hyperparameter value combination giving the highest weighted F1-score was chosen as optimal for the model.

The performance of the model fitted with the optimal values for the hyperparameters was tested by predicting the sources in the test data set. Again, because of the stochasticity in neural networks the final training and performance testing were repeated 30 times, and the average performance for the predictions of the test data set was calculated. The performance metrics used for evaluating the prediction performance were the F1-score for each source, the weighted F1-score, and the macro F1-score.

2.6.4 Predicting sources for isolates from clinical cases

The models for the different machine learning methods trained on the cgMLST training data set with all features and the wgMLST training data set were used to predict the sources for the isolates from the clinical cases of listeriosis filtered out during data cleaning. These isolates came from clinical human cases without known source for the infection. The trained machine learning models were used to try and attribute them to one of the sources the models were trained on. The results were used to compare the different machine learning methods used for the models and the two MLST methods used to represent the data.

3.0 Results

3.1 Preprocessing and data cleaning

The data consisting of the 1097 genomes for the isolates from the sources meat processing factories, salmon processing factories, rural and urban environments, dairy farms, slugs and clinical cases were run through the software tools ALPPACA and chewBBACA to make allelic profiles for the cgMLST and wgMLST data sets. After running the software tool ALPPACA with the external cgMLST schema on the data set, the cgMLST data set consisted of cgMLST profiles for the isolates. The software tool chewBBACA first constructed a schema for wgMLST consisting of 4923 loci. Then the allele calling done by chewBBACA made wgMLST profiles for the isolates and at the same time detected 32 paralogous genes. Further, chewBBACA was used to remove the paralogous genes. After removing these genes, the wgMLST data set consisted of the isolates with wgMLST profiles for 4891 loci.

Figure 16 shows a small illustration of a part of the cgMLST data set with one row for each isolate with a unique SRA number for identification and one column for each locus. The numbers in the cgMLST data set represent the allele types found in each isolate for each locus. Some isolates had a flag in some of the columns instead of a number, and the "LNF" indicated that no allele was found for that locus. The wgMLST data set was arranged in a similar fashion.

SRA_no	Pasteur_cgMLST-00023630	Pasteur_cgMLST-00023631	Pasteur_cgMLST-00023632	Pasteur_cgMLST-00023633
SRR13588210	8	1	22	427
SRR13588211	8	17	7	427
SRR13588213	1	1	7	433
SRR13588214	LNF	2	2	434
SRR13588215	LNF	2	2	434
SRR13588216	LNF	2	2	434
SRR13588217	LNF	2	2	434
SRR13588218	13	13	7	432
SRR13588219	LNF	2	2	434
SRR13588220	8	1	22	427
SRR13588221	8	1	22	427

Figure 16. Part of the output file for the cgMLST data set after the allelic profiles were made by ALPACCA. Each row is an isolate with a unique SRA number for identification, the columns are the names of the loci from the Institute Pasteur Listeria monocytogenes cgMLST schema, and the numbers represent the different allele types found at each locus for the isolates. Some of the isolates has the flag "LNF" for missing locus instead of the allele type number. These, and other flags were as described in the text further processed before analyses.

The output for the schema evaluation of the wgMLST schema gave an overview of the alleles and loci that were found but did not count any problematic alleles. The summary statistic for the evaluation is shown in Figure 17. The total number of loci and alleles are listed, and no invalid alleles were found. The "Total alleles not multiple of 3" counts alleles that are not complete because the sequence size are too small, but no threshold was set for size and the default is 0, "Total alleles w/>1 stop codon" counts alleles with more than one stop codon, "Total alleles wo/ start/stop codon" counts alleles missing start and stop codons, and "Total alleles shorter than 0 nucleotides" counts alleles shorter than the value set for minimum sequence length, but no minimum value was set and the default is 0.

Schema Summary Statistics

chewBBACA Version	Blast Score Ratio	Total Loci	Total Alleles	Total Alleles not multiple of 3	Total Alleles w/ >1 stop codons
3.1.0	0.6	4923	67351	0	0

Total Alleles wo/ Start/Stop Codon	Total Alleles shorter than 0 nucleotides	Total Invalid Alleles
0	0	0

Figure 17. Schema evaluation for wgMLST schema. Shown is the chewBACCA version that made the schema and the BLAST score ratio used to cluster alleles of the same locus. The total number of loci and alleles are listed. All columns for problems with the alleles show 0 alleles. "Total alleles not multiple of 3" indicates too short alleles based on a size threshold with default value 0, "Total alleles w/>1 stop codons" indicates alleles with more than one stop codon, "Total alleles wo/ start/stop codons" indicate alleles without start and/or stop codons, "Total alleles shorter than 0 nucleotides", indicates alleles shorter than minimum sequence length with the default value 0, "Total invalid alleles" indicate the total number of alleles found invalid.

The first cleaning steps removed flags and replaced the flags for missing alleles with "NaN"-values in both data sets. Then the isolates with the source clinical cases were taken out of the data sets, and thus the number of isolates per data set were reduced to 967 isolates. Filtering based on missing values were used to prepare the data sets for machine learning. For the cgMLST data set 14 loci were filtered out because 10% or more of the isolates had missing values for the locus. After this filtering the cgMLST data set consisted of 1734 columns of loci as features. In the wgMLST data set 2395 loci were filtered out using the same cutoff. After the filtering the wgMLST data set consisted of 2496 columns of loci as features.

The threshold for filtering out sources with number of isolates below 15 did not filter out any sources for any of the data sets since no isolates were filtered out during cleaning and all sources originally were above 15 isolates per source.

3.2 Data exploration

3.2.1 Differences in MLST methods and diversity of data

When the allelic profiles for the cgMLST data set and the wgMLST data set were made two different schemas were used, and exploration was done to get an insight to how different the two schemas gave number values to the distinct alleles. The 5 highest number values given to alleles were recorded and compared for both data sets. Also, for each locus the number of distinct alleles was counted, and the highest number counted in each data set was recorded.

The highest number of distinct alleles counted among the loci in the cgMLST data set was 55, for the wgMLST data set it was 77. The 5 highest number values for the allele types ranged from 729 to 863 for the cgMLST profiles and 62 to 92 for the wgMLST profiles. The highest number values for alleles in the two data sets is recorded in Table 4.

Table 4. The 5 highest number values assigned to allele types for the cgMLST and wgMLST data sets.

Data set	**The 5 highest number values assigned to allele types**				
cgMLST	863	860	773	731	729
wgMLST	92	69	63	62	60

The diversity was explored by calculating Shannon entropies for the loci in the cgMLST and the wgMLST data sets. A bar plot of the distribution of the Shannon entropies among the loci for both data sets is presented in Figure 18. The lowest entropy calculated was 0 for both data sets, and the highest was 2.96 for the cgMLST data set and 3.10 for the wgMLST data set. The average values were 1.86 for the cgMLST data set and 1.89 for the wgMLST data set.

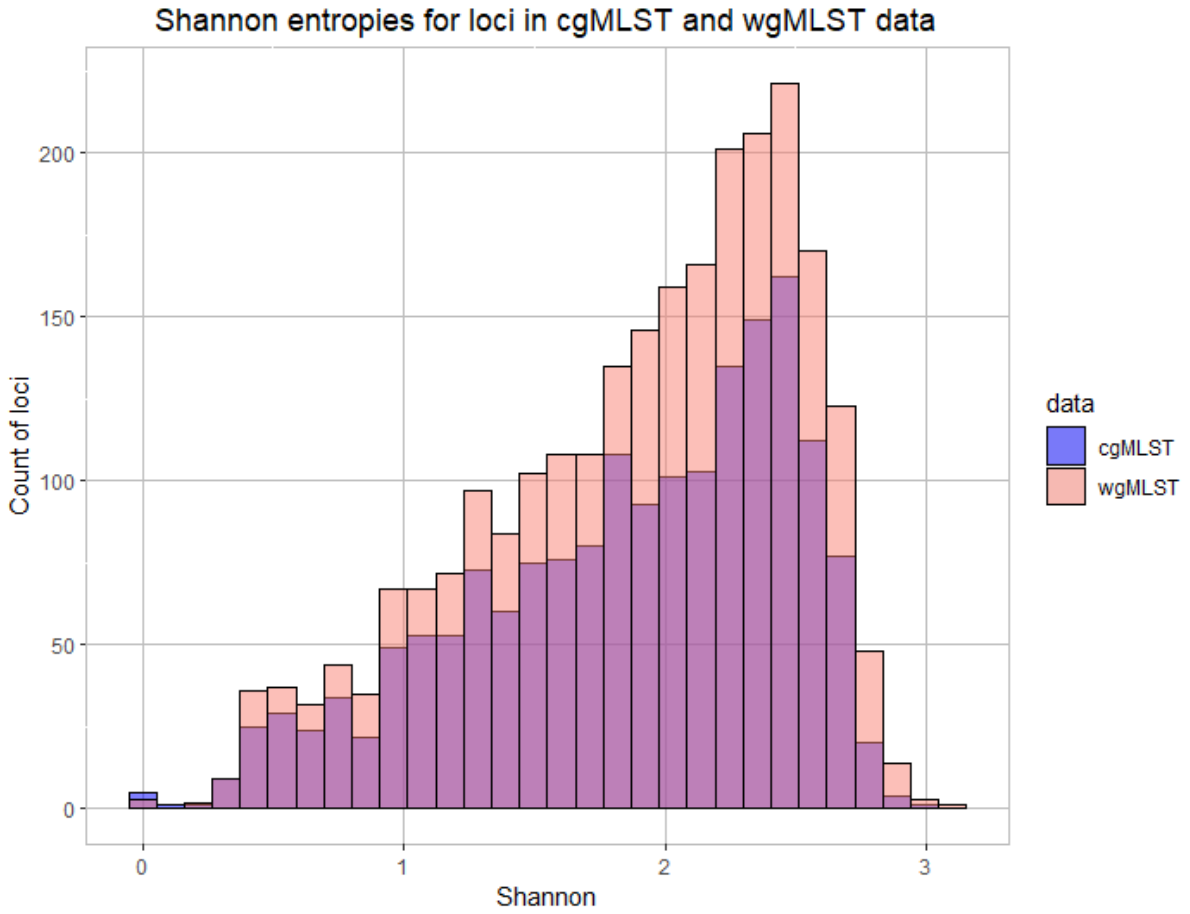

Figure 18. Distribution of Shannon entropies for loci in the cgMLST and wgMLST data sets.

3.2.2 Clustering

The cgMLST and the wgMLST data sets were clustered using two different unsupervised machine learning methods to look for grouping among the isolates within each data set. The clustering was first performed using k-mode clustering. The number of clusters the model tried to find was 5 to look for natural clustering of the 5 sources in the data sets. The distribution of the sources within the clusters for the cgMLST data set is shown in Figure 19. Cluster 3 was the cluster most clearly dominated by one source. This cluster had the source meat processing factory for 97% of the isolates it contained. The cluster with the second highest presents of isolates coming from the same source was cluster 2, where 69% of the isolates had salmon processing factory as source. In the other clusters the isolates sources were more of a mix.

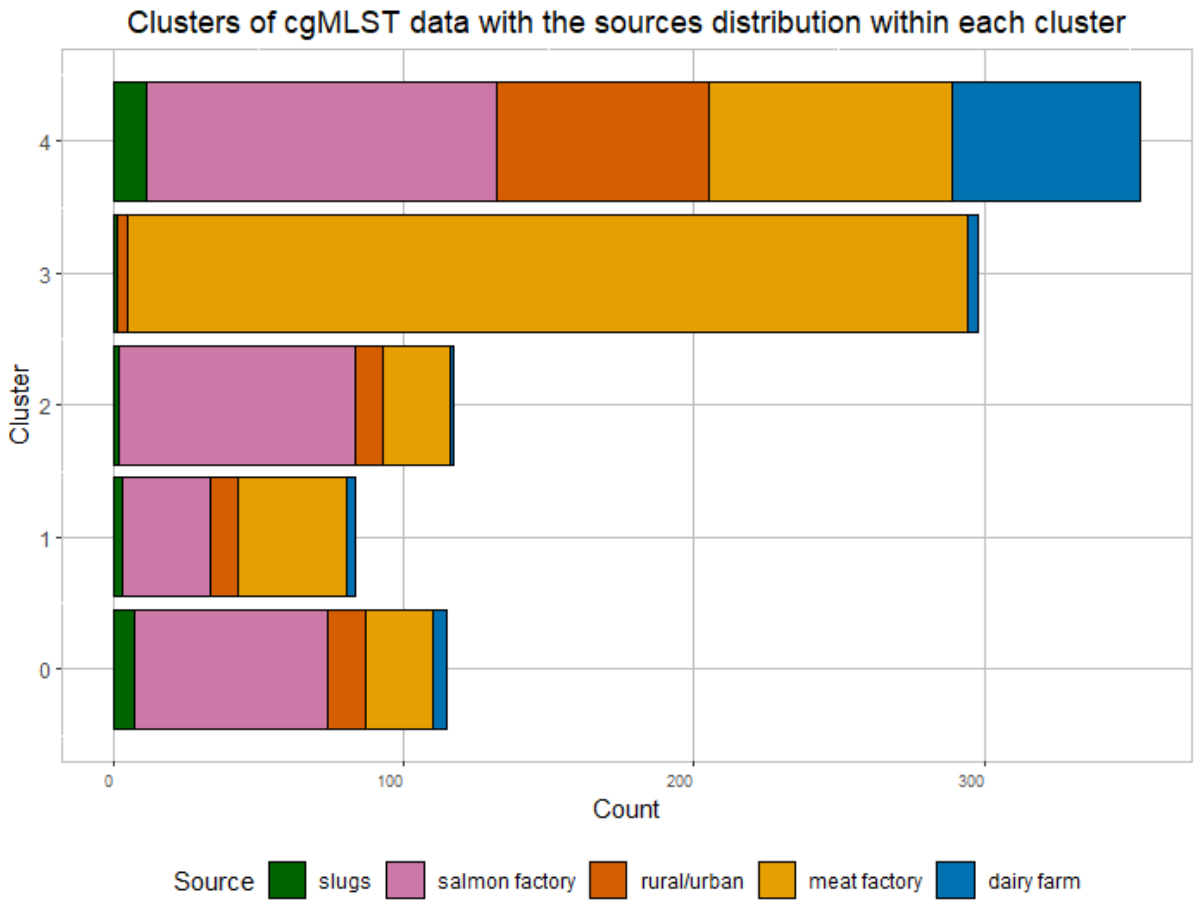

Figure 19. K-modes for the cgMLST data set with 5 clusters and the sources distribution within them.

The distribution of the sources within the clusters for the wgMLST data set is shown in Figure 20. The wgMLST data set showed very much the same clustering as the cgMLST data set. Also here cluster 3 was dominated by one source, and 91.8% of the isolates in the cluster had the source meat processing factory. Cluster 2 had the second highest presents of isolates coming from the same source with 72.3% of isolates with the source salmon processing factory.

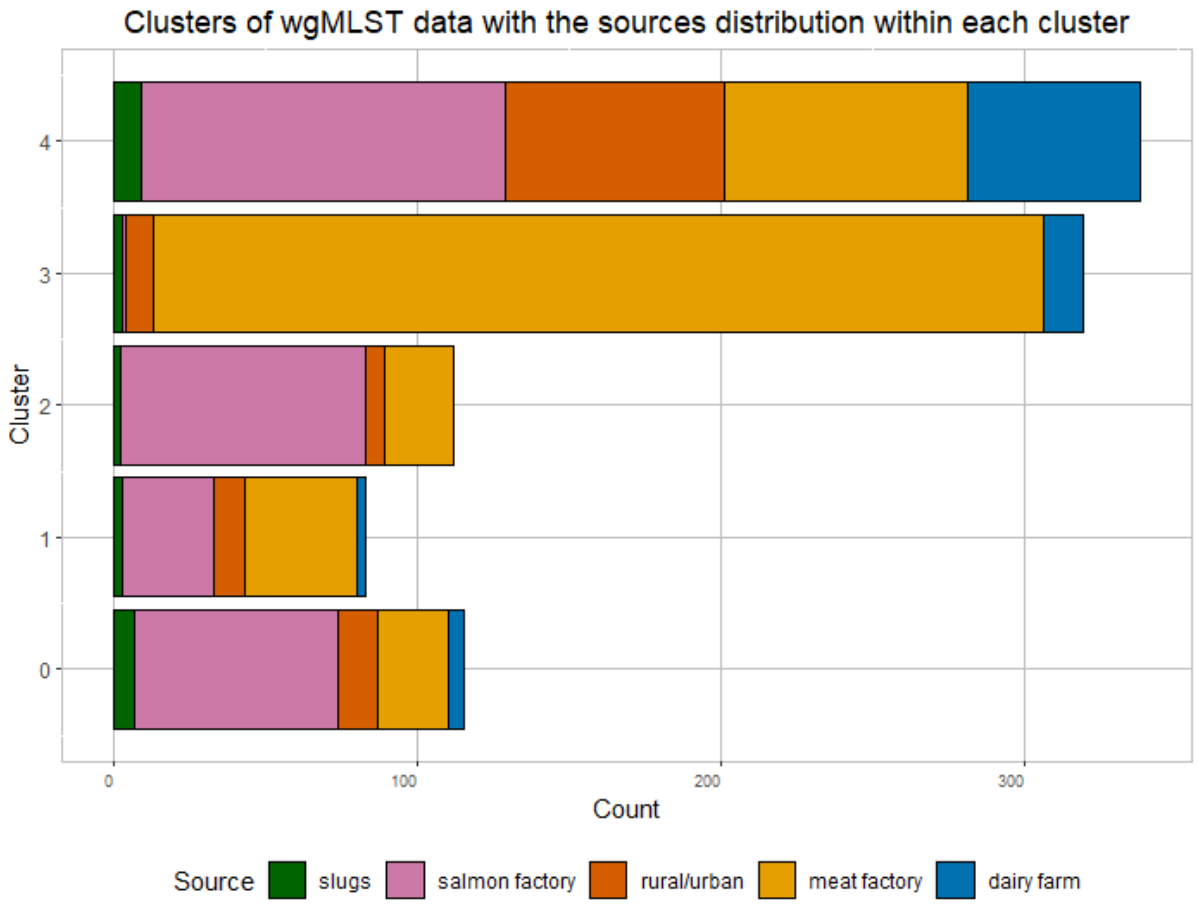

Figure 20. K-modes for the wgMLST data set with 5 clusters and the sources distribution within them.

Hierarchical clustering was also performed on the cgMLST and the wgMLST data sets. The clustering was performed using Hamming distances, and a dendrogram of the clustering together with a heatmap for the distances is presented in Figure 21 for the cgMLST data set. The hierarchical clustering and heatmap showed a group of meat processing factory isolates with high similarity presented as the largest black square in the heatmap. There was one smaller group of mostly isolates with salmon processing factory as source displaying some similarity. The rest of the isolates clustering and showing signs of similarities were a mix of sources.

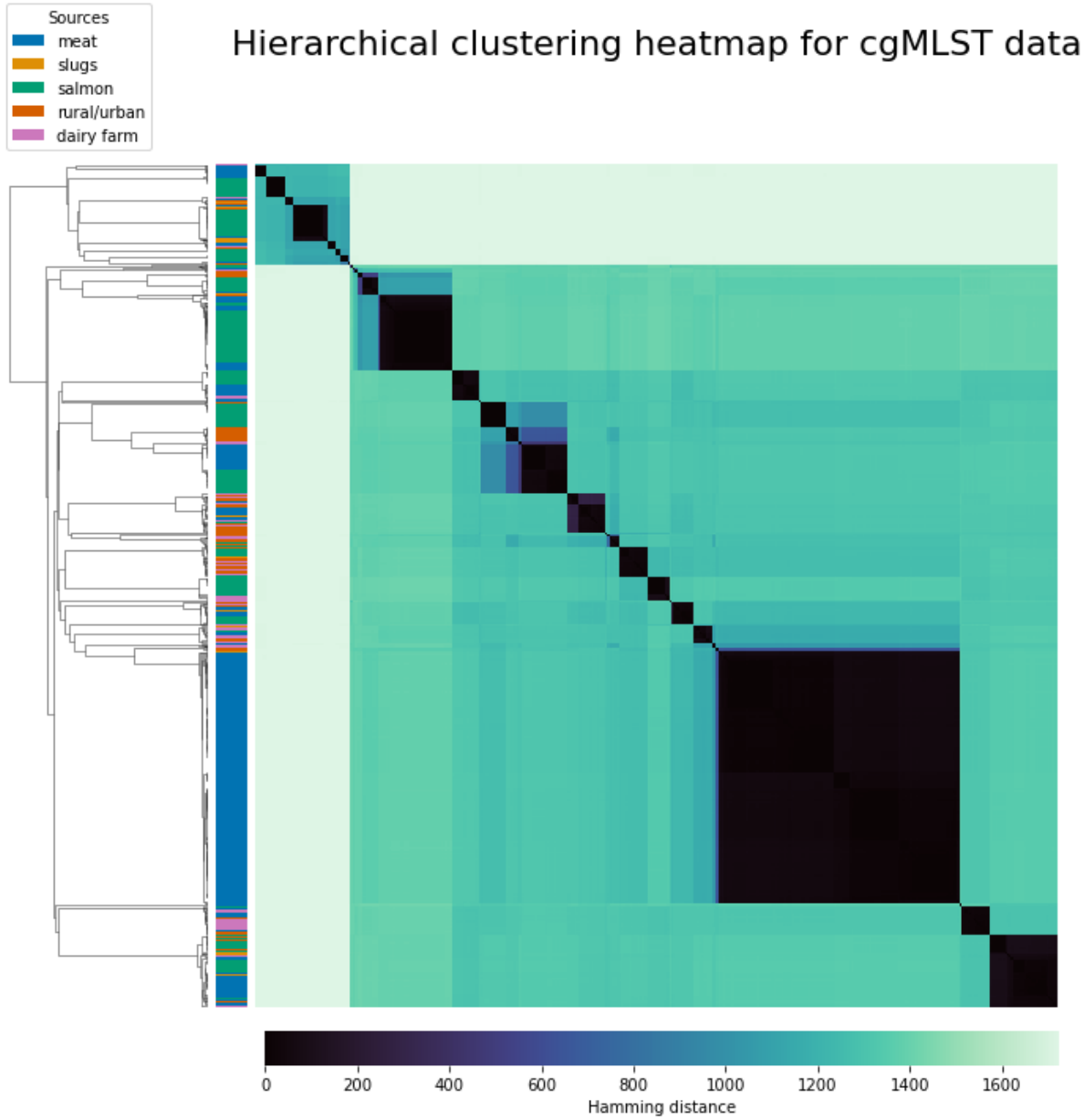

Figure 21. Clustering using the cgMLST data set showing a dendrogram of hierarchical clustering, with sources for the clustered isolates shown by colour on the side, and a heatmap displaying the Hamming distances between the isolates.

A dendrogram of the clustering together with a heatmap for the distances for the wgMLST data set is presented in Figure 22. The hierarchical clustering and heatmap of the wgMLST data set also showed a group of meat processing factory isolates with high similarity presented as the largest black square in the heatmap, and one smaller group of mostly salmon processing factory isolates displaying some similarity in this plot.

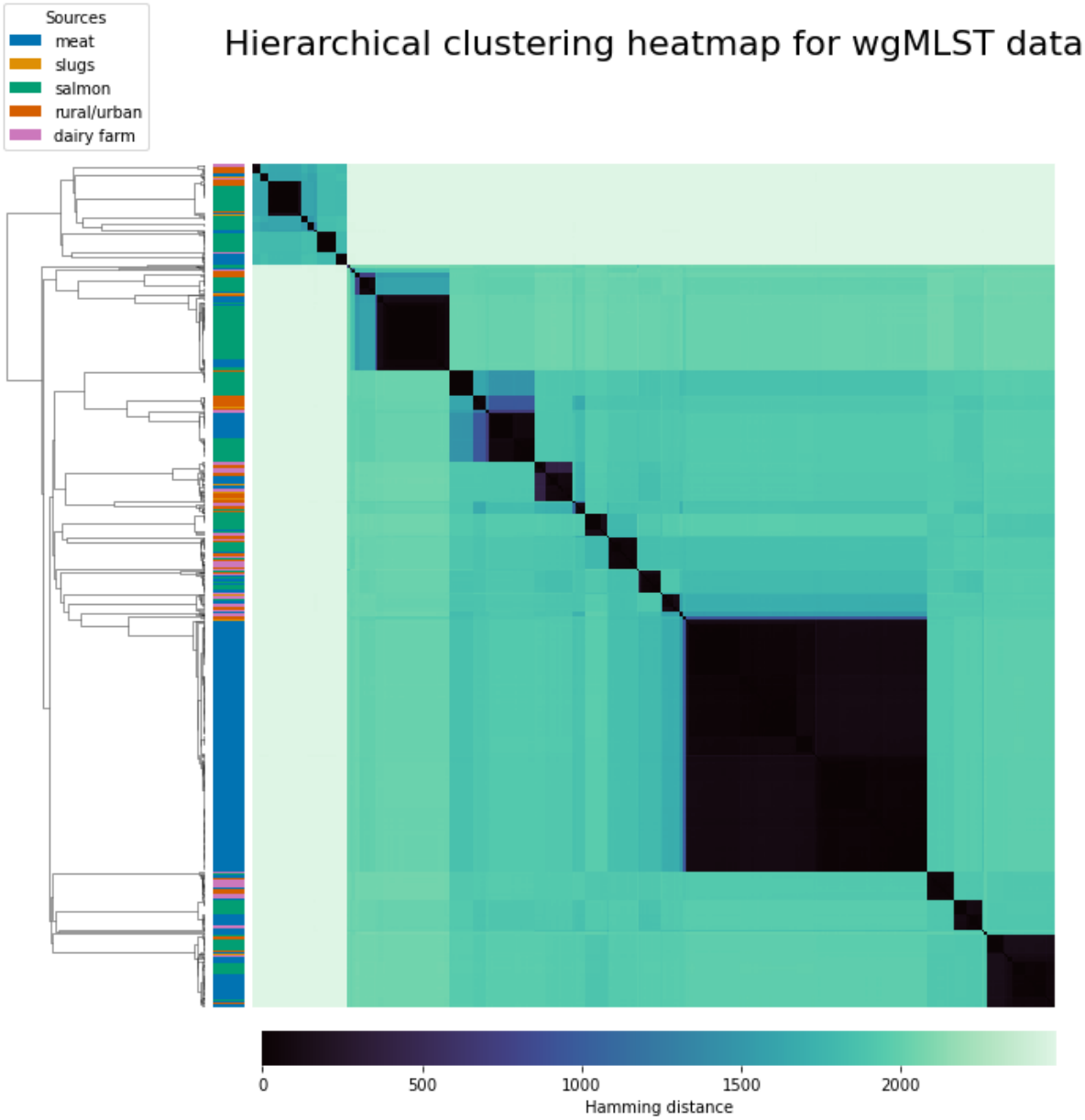

Figure 22. Clustering using the wgMLST data set showing a dendrogram of hierarchical clustering with sources for the clustered isolates shown by colour on the side, and a heatmap displaying the Hamming distances between the isolates.

3.3 Feature-selection

Before feature-selection the cgMLST and the wgMLST data sets were split into training data and test data sets in a ratio of 70/30. After the split the training data sets consisted of 676 isolates, and the test data sets consisted of 291 isolates.

3.3.1 Mutual information

Using only the cgMLST training data set the mutual information was calculated for each feature with the class variable holding the source labels giving a score for each feature. A bar plot of the distribution of mutual information scores among the 1734 features is shown in Figure 23. The lowest mutual information score was 0, and the highest 0.70. The average mutual information score of the features was 0.36. The higher the mutual information was calculated to be, the more information the feature shared with the class variable.

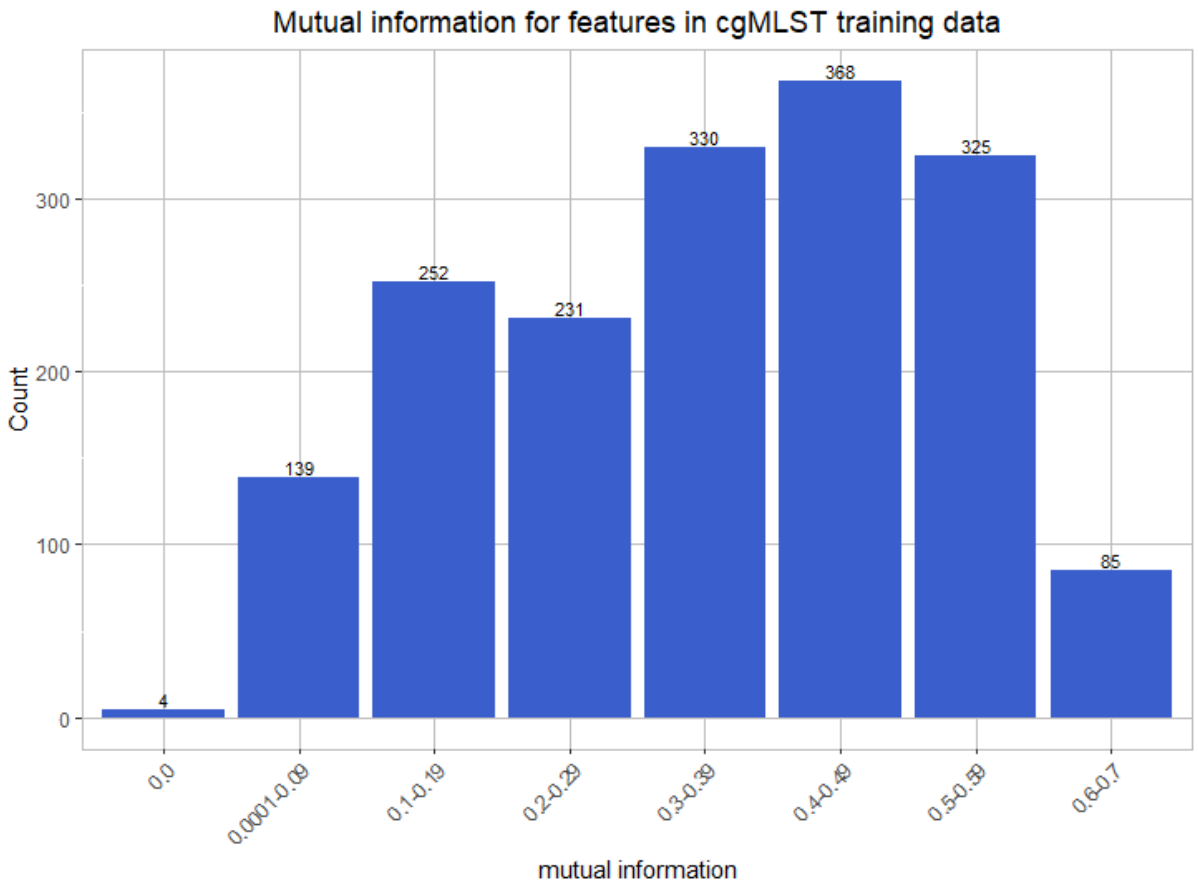

Figure 23. Distribution of the features mutual information score in the cgMLST training data set.

The 10, 20, 30, 40 and 50 percent of the features with the highest mutual information scores were selected as separate cgMLST training data sets and used further to compare the performance of the machine learning models based on the smaller subsets of the features in the cgMLST data set. Table 5 shows the number of features in the feature selected data sets along with the range of their mutual information score compared with the cgMLST data set

before feature-selection. All the feature selected cgMLST data sets had features with mutual information scores above the average of 0.36.

Table 5. Feature selected cgMLST data sets compared to the cgMLST data set with all features. Showing percent, number of features and range of mutual information score for the features in the data sets.

Percent of features	Number of features	Range of mutual information score among the features
10	174	0.57 - 0.70
20	347	0.52 - 0.70
30	520	0.47 - 0.70
40	694	0.42 - 0.70
50	867	0.37 - 0.70
All cgMLST	1734	0 - 0.70

By comparing the 10 percent of the loci with the highest Shannon entropies calculated earlier, with the 10 percent of the loci used as features with the highest mutual information score, it was found that 142 of the 174 loci with highest entropies also were found among the features with the 10 percent highest mutual information scores.

3.4 Machine learning models for prediction

3.4.1 Training models

Optimal model hyperparameters:

Different models were made by training them on the different training data sets and optimising the hyperparameters with a grid search using cross validation. The optimal hyperparameters found by the grid search are summed up in Table 6 for all models. The cross validation in the grid search calculated average weighted F1-scores. The optimal hyperparameter or combinations of hyperparameters, depending on the method for the model, were the ones giving the highest average weighted F1-score.

Table 6. Random Forest, Support Vector Machine with and without scaling, and neural network models optimal hyperparameters for models trained before and after feature-selection on the cgMLST training data set and on the wgMLST training data set.

Percent Features/ Training data set	**Random Forest** (no. decision trees)	**Support Vector Machine with scaling** (Cost and gamma)	**Support Vector Machine without scaling** (Cost and gamma)	**Shallow dense neural network** (Nodes, dropout rate, learning rate)
10% cgMLST	No. trees=600	Cost=4.0, Gamma= 0.002	Cost =1.5, Gamma =0.0005	Nodes =80, Dropout =0.2 Learning =0.001
20% cgMLST	No. trees =600	Cost =5.0, Gamma = 0.001	Cost =2.0, Gamma =0.0005	Nodes =80, Dropout =0.2 Learning =0.001
30% cgMLST	No. trees =800	Cost =3.0, Gamma = 0.001	Cost =1.5, Gamma =0.0005	Nodes =80, Dropout =0.2 Learning =0.001
40% cgMLST	No. trees =800	Cost =4.0, Gamma = 0.001	Cost =1.5, Gamma =0.0005	Nodes =75, Dropout =0.2 Learning =0.001
50% cgMLST	No. trees =500	Cost =2.0, Gamma = 0.001	Cost =1.5, Gamma =0.0005	Nodes =75, Dropout =0.2 Learning =0.001
All cgMLST	No. trees =400	Cost =2.0, Gamma =0.0005	Cost =1.5, Gamma =0.0005	Nodes =80, Dropout =0.2 Learning =0.0001
All wgMLST	No. trees =300	Cost =5.5, Gamma =0.005	Cost =3.0, Gamma =0.0005	Nodes =80 Dropout =0.2 Learning =0.0001

The number of decision trees found optimal by the grid search for the Random Forest models ranges from 300-800. The time for running the grid search and finding the optimal hyperparameter for a Random Forest model was between 2 to 6 minutes depending on the number of features in the training data set.

For the Support Vector Machine models with scaling, the optimal hyperparameter value determined by the grid search for the cost ranged from 2.0-5.5, and for the gamma it ranged from 0.0005-0.005. The time for running the grid search and finding the optimal hyperparameter combination for one model was from 1 to 10 minutes depending on the number of features in the training data set.

For the Support Vector Machine models without scaling, the optimal hyperparameter values determined by the grid search for the cost were 1.5 for all models except two. The model trained on the feature selected cgMLST training data set with 20 percent of the features had the value 2.0 for the cost hyperparameter, and the model trained on the wgMLST training data set had the value 3.0. The optimal value for the hyperparameter gamma was 0.0005 for all models. The time for running the grid search and finding the optimal hyperparameter combination for one model was from 1 to 11 minutes depending on the number of features in the training data set.

For the shallow dense neural network, the optimal hyperparameter values determined by the grid search for the number of nodes was either 75 or 80, and the dropout rate was 0.2 for all models. The learning rate for the optimiser was 0.001 for all the models trained on feature selected cgMLST training data sets, and 0.0001 for the model trained on the cgMLST training data set with all features and the same for the wgMLST training data set. The grid search was run 5 times for the shallow dense neural network models, and one run took from 10-15 minutes making the total time for one model 50-75 minutes depending on the number of features in the training data set.

Training performance:

The training performance for the models fitted with the optimal hyperparameters are here defined as the average F1-score calculated by the cross validation in the grid search for the optimal hyperparameters, and they are shown in Figure 24. If the model always predicts correct sources for all isolates during cross validation the weighted F1-score will be 1.0, so the higher the score the better the model is at predicting. The plot in Figure 24 shows the models training performance between the different machine learning methods and MLST methods to represent the data, and across the different number of features in the training data sets.

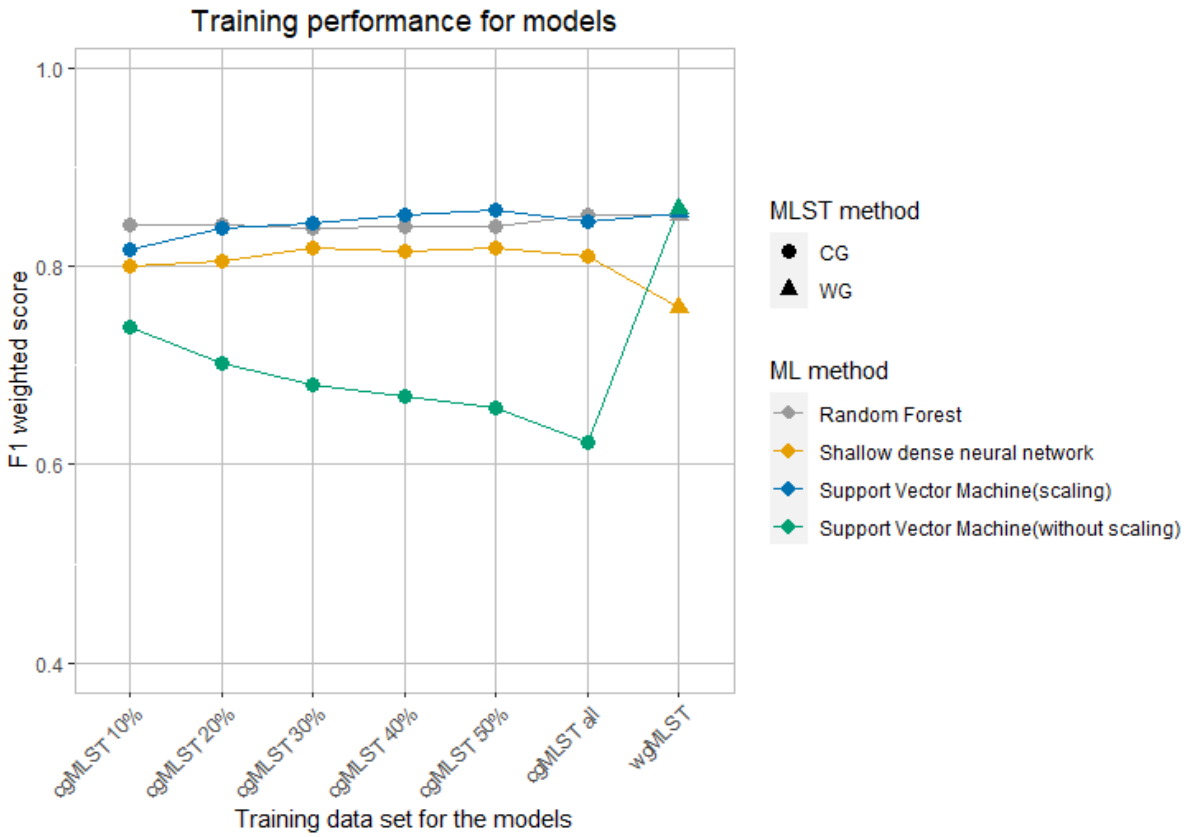

Figure 24. Results for training performance for the models. The x-axis shows the different training data sets with different number of features. The y-axis is the value of the weighted F1-score calculated by the cross validation for the model with its optimal hyperparameters. The different colours represent the different machine learning methods, the Support Vector Machine is represented by both models with scaling of features and without, and the shape of the point indicates the MLST method used to make the allelic profiles of the data.

A seed was set for reproducibility in the function for the Random Forest models, and different seeds were tested to see if this had impact on the performance. The training performance of the Random Forest models did not alter noticeably by changing the seed. The highest training performance for the Random Forest method had a weighted F1-score of 0.851. This was the performance of both the model trained on the cgMLST training data set with all features and the model trained on the wgMLST training data set. The training performance of the Random Forest models trained on cgMLST feature selected training data sets all had a weighted F1-score within the range 0.838-0.842. The training performance of the Random Forest models were stable with only small variation across the different training data sets as shown in Figure 24.

For the Support Vector Machine method with scaling, the highest training performance had a weighted F1-score of 0.856. This was for the model trained on the feature selected cgMLST training data set with 50 percent of the features. The model trained on the cgMLST training data set with all features and the rest of the models trained on the cgMLST feature selected training data sets had a training performance in the range of 0.817-0.851. The training performance of the model trained on the wgMLST training data set was 0.854. Apart from slightly lower performance for the model trained on the cgMLST feature selected data set with 10 percent of the features, Figure 24 shows a stable training performance across the different training data sets for the Support Vector Machine models with scaling.

The highest training performance for the Support Vector Machine method without scaling, was a weighted F1-score of 0.858. This was the training performance of the model trained on the wgMLST training data set, and the performance was notably higher than the other models. The range for the weighted F1-scores for all the models trained on cgMLST training data sets was 0.623-0.739. The trend seen in Figure 24 shows a declining training performance for the models as the number of features increases in the cgMLST training data sets, before going up again for the model trained on the wgMLST training data set.

For the shallow dense neural network, the highest training performance was a weighted F1-score of 0.818. This was the score for both the model trained on the cgMLST feature selected data sets with 30 percent of the features and 50 percent of the features. All the models trained on cgMLST training data sets had a training performance with a weighted F1-score in the range 0.801-0.818. The training performance drops down to a weighted F1-score of 0.760 for the model trained on the wgMLST training data set. The trend in Figure 24 shows a stable training performance for the models trained on cgMLST training data sets, but a drop in performance for the model trained on the wgMLST data set.

3.4.2 Evaluating model predictions

The trained models fitted with their optimal hyperparameters were used to predict the sources for the isolates in the test data set. The performances of the predictions were evaluated by calculating the F1-score for each source along with the weighted F1-score and macro F1-

score. Confusion matrices were also made for the models trained on the cgMLST training data set with all features and the wgMLST training data set to compare how well the models trained on the different MLST profiles distinguish between the sources.

Confusion matrices:

Figure 25 shows the confusion matrices for the predictions of the isolates in the test data set's sources done by the different machine learning methods with models trained on all the features in the cgMLST training data set. Because of the imbalance of isolates per source in the data the matrices display the fraction of predicted isolates per total number of isolates in the true source. The Random Forest model had a high rate of correct predictions for both the isolates from meat processing factories and the salmon processing factories. The isolates with sources dairy farms, and rural and urban environments had more incorrectly predicted isolates than the meat and salmon processing factories. However, around half of the predictions were correct for the isolates from dairy farms, and more for the rural and urban environments. The isolates from dairy farms were most frequently incorrectly predicted to have rural and urban environments as source, and the isolates from rural and urban environments were most frequently incorrectly predicted to have salmon processing factory as source. The isolates from slugs were incorrectly predicted more than they were correctly predicted, and they were predicted incorrectly as isolates from all the other sources. The predictions of the Random Forest model, the shallow dense neural network, and the Support Vector Machine with scaling were very similar. The Support Vector Machine without scaling, predicted mostly meat processing factory as source for all isolates and had a high rate of incorrect predictions.

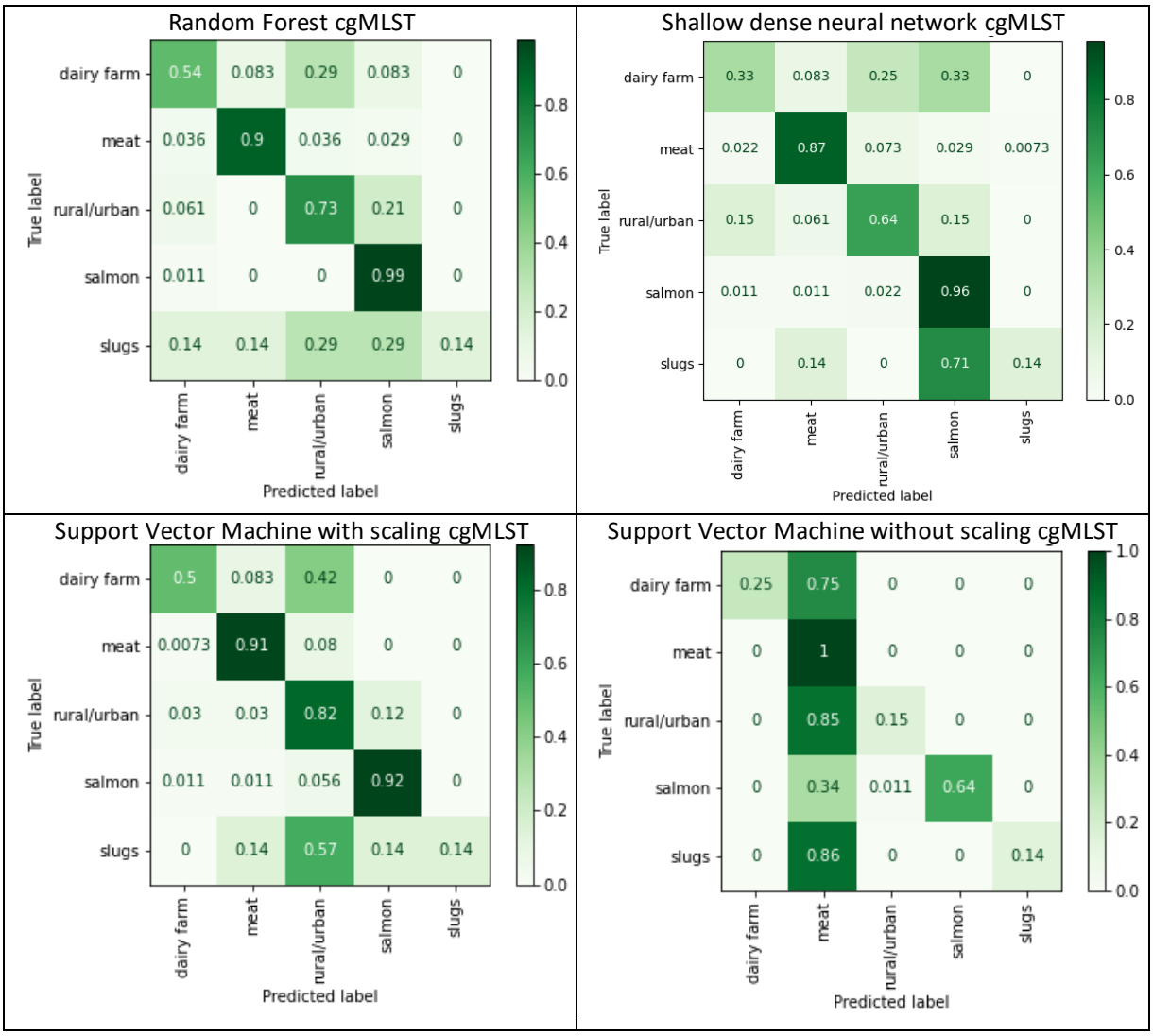

Figure 25. Confusion matrices for the predictions of the isolates in the test data set done by the models trained with all features in the cgMLST data set. The matrices show the fraction of predictions per source per the total number of isolates in each true source. There is one confusion matrix for each of the models Random Forest, shallow dense neural network, Support Vector Machine with scaling, and Support Vector Machine without scaling. The true sources of the isolates are the rows and the model's prediction are the columns in the matrix.

Figure 26 shows the confusion matrices for the predictions of the source for the isolates in the test data set done by the different machine learning methods with models trained on the wgMLST training data set. Also for these matrices the predicted source for the isolates in the matrices are displayed as the fraction of predicted isolates per total number of isolates in the true source. Adding more features and changing the MLST method did not alter the predictions much for the Random Forest model, the shallow dense neural network model, or the Support Vector Machine model with scaling. However, the predictions for the Support Vector Machine model without scaling changed to the better when the model was trained on

the wgMLST training data set compared to the cgMLST training data set. The model managed to correctly predict most of the isolates from meat and salmon processing factories, and rural and urban environments. Around a third of the isolates with dairy farms as source were also predicted correctly, and two thirds were incorrectly predicted as isolates from rural and urban environments. Like the other models, this model could not predict the correct source for the isolates from slugs either.

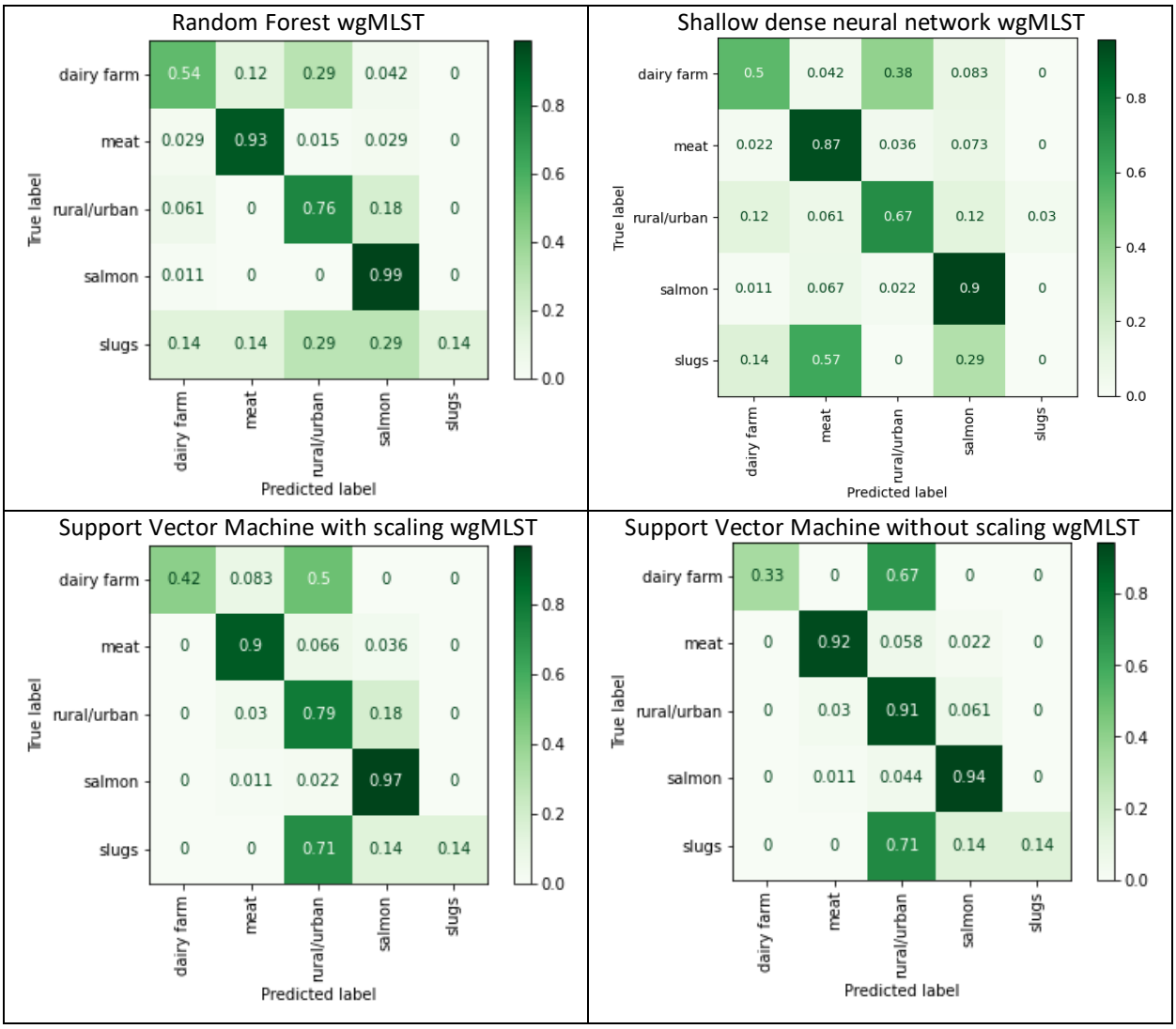

Figure 26. Confusion matrices for the predictions of the isolates in the test data set done by the models trained with the wgMLST training data set. The matrices show the fraction of predictions per source per the total number of isolates in each true source. There is one confusion matrix for each of the models Random Forest, shallow dense neural network, Support Vector Machine with scaling, and Support Vector Machine without scaling. The true sources of the isolates are the rows and the model's prediction are the columns in the matrix.

Test performance per source:

The plots in Figure 27 shows the different models ability to predict the source for the isolates in the test data set per source. The plots show the F1-scores calculated per source for each of the machine learning models trained on the different number of features and with allelic profiles from the different MLST methods. The Random Forest, the shallow dense neural network, and the Support Vector Machine method with and without scaling each have one plot displaying the F1-scores for their models per source in the Figure.

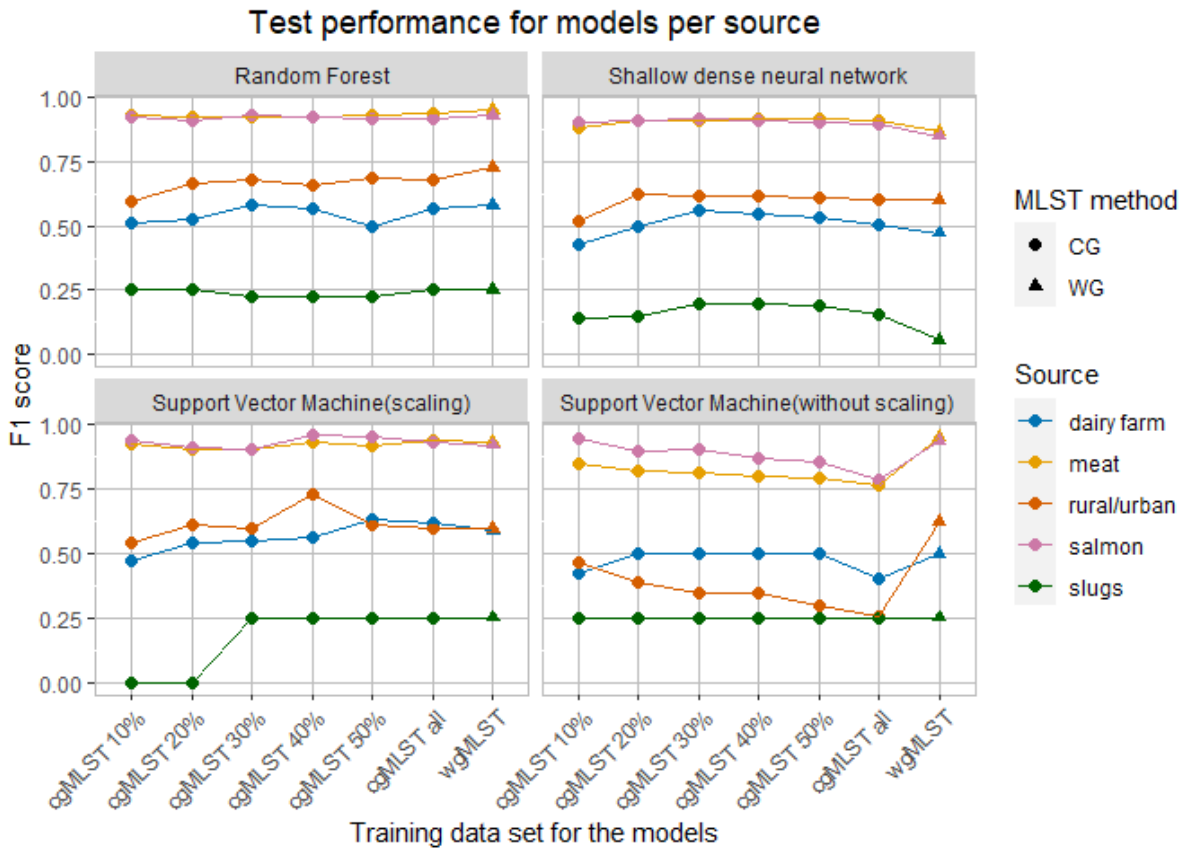

Figure 27. Test performance results with F1- score per source after the models predicted the sources for the isolates in the test data set. There is one plot for each machine learning method, the Support Vector Machine is represented by both the models with scaling and without. The x-axis shows the different training data sets with different number of features, and the shape of the point indicates the MLST method used to make the allelic profiles of the data. The colours represent the different sources.

The Random Forest models had high performance scores for isolates from the food-associated sources meat and salmon processing factory, the F1-scores for the models were in the range

0.920-0.948 for the isolates from meat processing factories, and 0.912-0.930 for salmon processing factories. For the isolates with rural and urban environments as source the F1-scores for the models were in the range 0.597-0.725, and for the isolates with dairy farms as source the range was 0.500-0.579. For the isolates with slugs as source the F1-scores for the models were low and in the range 0.222-0.250.

For the Support Vector Machine with scaling, the highest F1-scores were for the isolates from the food-associated sources, meat and salmon processing factory. The range of the F1-scores for the models was 0.904-0.936 for the source meat processing factory, and 0.904-0.960 for the source salmon processing factory. For the isolates with rural and urban environments as source the models had their F1-scores in the range 0.540-0.727. The F1-score for the predictions of the isolates with dairy farms as source were in the range 0.471-0.634. For the isolates with slugs as the source the F1-scores were low and equal to 0 for both the model trained on the feature selected cgMLST training data sets with 10 percent features and 20 percent features. The rest of the models had the F1-score 0.250.

For the Support Vector Machine without scaling, the F1-scores for the isolates with meat processing factories as source ranged from 0.768-0.951 for the models. For the isolates with salmon processing factories as source, the F1-scores ranged from 0.784-0.943. For the isolates with rural and urban environments as source, the F1-scores ranged from 0.256 -0.625 for the models. The F1-score for the isolates with dairy farm as source were 0.500 for all models apart from the model trained on the feature selected cgMLST training data set with 10 percent features which had the score 0.424, and the model trained on the cgMLST training data set with all features which had the score 0.400. The isolates with slugs as source had stable but low predicting performance with the F1-score 0.25 for all models.

The F1-scores for the shallow dense neural network models also had highest scores for the isolates from the food-associated sources. For meat processing factories the models F1-scores were in the range 0.865-0.915, and 0.848-0.912 for isolates from salmon processing factories. For the isolates with rural and urban environments as source the models had F1-scores in the range 0.518-0.623. For the isolates with dairy farms as source the F1-scores were in the range

0.428-0.557 for the models. The predictive performance for the isolates with slugs as source were low, and the F1-scores for all the models were in the range 0.056-0.200.

A table with all the test performances for the models as F1-scores per source can be viewed in Appendix A.

Average test performance:

A plot of the weighted F1-scores and the macro F1-scores for the models are visualised in Figure 28. The plot shows the comparison of the predictive performances of the models on the test data sets between machine learning methods, and across the different numbers of features in the training data sets and allelic profiles from the different MLST methods.

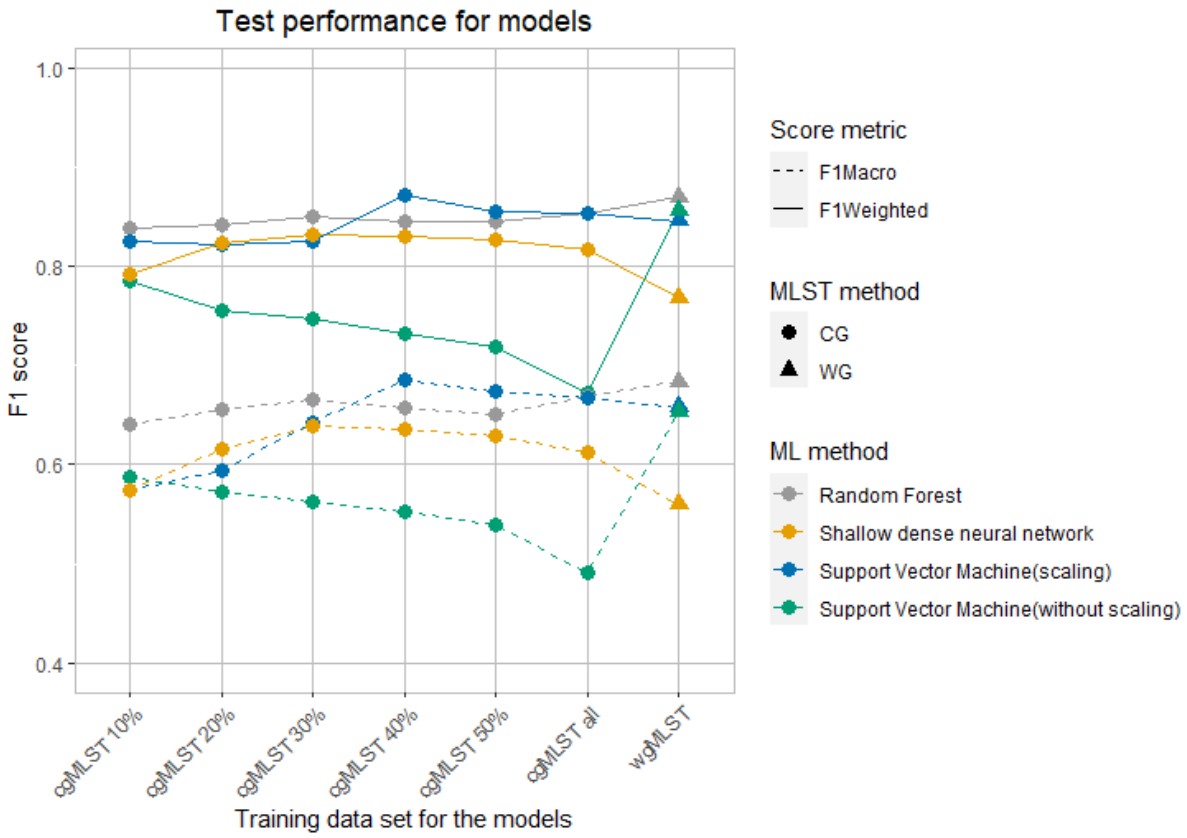

Figure 28. Test performance results with F1- weighted and F1-macro scores after the models predicted the sources for the isolates in the test data set. The x-axis shows the different training data sets with different number of features. The y-axis is the value of the weighted/macro F1-score calculated for the performance. The solid line shows the weighted F1-scores, and the dashed line shows the macro F1-scores. The different colours represent the different machine learning methods, the

Support Vector Machine is represented by both the models with scaling and without, and the shape of the point indicates the MLST method used to make the allelic profiles of the data.

The Random Forest method and the Support Vector Machine with scaling had the highest performance scores for the different models. The Random Forest method had the most similar performance between the models, and the Support Vector Machine model with scaling trained on the feature selected cgMLST training data set with 40 percent of the features had the highest recorded test performance, with a weighted F1-score of 0.871 and a macro F1-score of 0.686.

The highest test performance for the Random Forest method was for the model trained on the wgMLST training data set and had a weighted F1-score of 0.869 and a macro F1-score of 0.685. The test performance of the Random Forest models trained on the cgMLST training data sets before and after feature-selection all had a weighted F1-scores within the range 0.838-0.853 and a macro F1-score within the range 0.641-0.669. The test performance of the Random Forest method was stable with only small variation across the models trained on the different number of features in the cgMLST training data sets and had a peak in performance for the model trained on the wgMLST training data set as shown in Figure 28.

For the Support Vector Machine method with scaling the highest test performance was for the model trained on the feature selected cgMLST training data set with 40 percent of the features, and it had a weighted F1-score of 0.871 and a macro F1-score of 0.686. The rest of the models trained on the different cgMLST training data sets had a test performance in the range of 0.822-0.855 for the weighted F1-score and a macro F1-score within the range 0.575-0.674. The model trained on the wgMLST training data set had a weighted F1-score of 0.846 and a macro F1-score of 0.658. Figure 28 shows that the test performance for the Support Vector Machine method with scaling had some variations in the performance for the different models with a peak in performance for the model trained on the feature selected cgMLST training data set with 40 percent of the features.

The highest test performance for the Support Vector Machine method without scaling was a weighted F1-score of 0.856 and a macro F1-score of 0.653. This was the test performance of the model trained on the wgMLST training data set, and the performance was notably higher than for the other models. The test performances for the models trained on the different cgMLST training data sets were in the range of 0.672-0.786 for the weighted F1-score and 0.492-0.587 for the macro F1-score. The trend seen in Figure 28 shows a declining test performance for the models as the number of features increases for the models trained on cgMLST training data sets.

For the shallow dense neural network, the highest test performance had a weighted F1-score of 0.832 and a macro F1-score of 0.639. This was the score for the model trained on the feature selected cgMLST training data set with 30 percent of the features. The rest of the models trained on the different cgMLST training data sets had a test performance with the weighted F1-scores in the range 0.791-0.830 and a macro F1-score in the range 0.574-0.636. The test performance drops down to a weighted F1-score of 0.778 and a macro F1-score of 0.560 for the model trained on the wgMLST training data set. The trend in Figure 28 shows a stable test performance across the models with a slight drop in performance for the model trained on the feature selected cgMLST training data set with 10 percent features and a drop for the model trained on the wgMLST training data set. The test performances for the shallow dense neural networks were an average of performance after predicting the sources for the isolates in the test data set 30 times. The average standard deviation for the weighted F1-scores was 0.020, and the average standard deviation for the macro F1-score was 0.036. These were the averages calculated for all the models using the shallow dense neural network as method.

A table with all weighted and macro F1-scores for the test performance of the models can be viewed in Appendix B.

3.4.3 Predicting sources for isolates from clinical cases

The sources for the isolates from the 130 clinical human cases of listeriosis were predicted using the models for the different machine learning methods that were trained on the cgMLST

training data set with all features and the wgMLST training data set. The distribution of prediction per source is presented in Figure 29. The full list of prediction for each isolate is found in Appendix C.

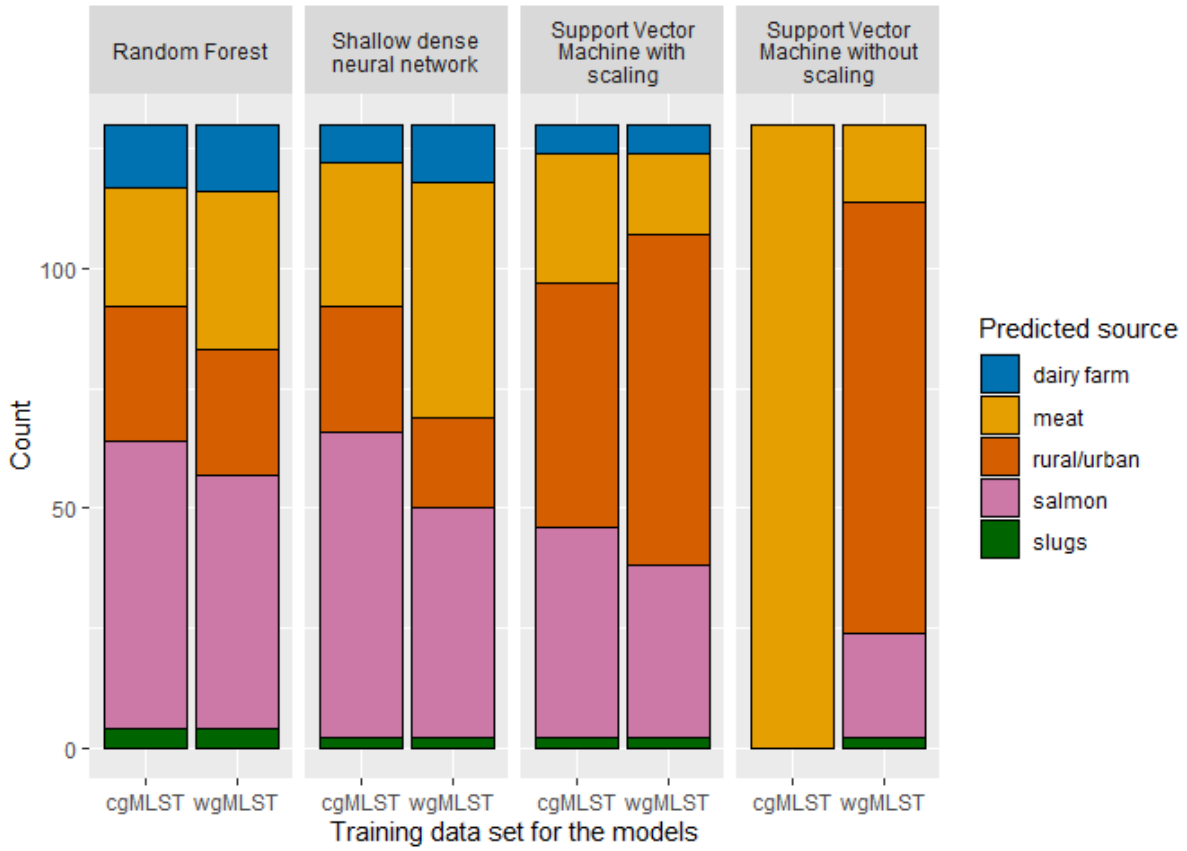

Figure 29. The predicted sources for the 130 isolates from clinical human cases of listeriosis. The distribution of prediction per sources is shown for the models trained on the cgMLST training data set with all features and the wgMLST training data set. The models are grouped by machine learning method, and the Support Vector Machine is represented both by models with and without scaling. The colours indicate the predicted source.

Figure 29 shows that all models except the Support Vector Machine models without scaling predicted at least a few isolates from the clinical cases to come from all the different sources. The Support Vector Machine model without scaling trained on the cgMLST training data set predicted all clinical isolates to have meat processing factories as source, and the model trained on the wgMLST training data set did not predict any clinical isolates to have dairy farm as source.

4.0 Discussion

The main aim of this study was to find out how supervised machine learning models could be used to predict the source of a set of Norwegian *Listeria monocytogenes* isolates by using their genomic data. Several models using different machine learning methods and two different methods for making MLST allelic profiles to represent the data were used to compare the predictive power between the different models. The three machine learning methods Random Forest, Support Vector Machine and a neural network were used to train models on cgMLST profiles before and after feature-selection and on wgMLST profiles. The secondary object for the study was to investigate the information held by the data sets using the diversity measure Shannon entropy and groupings of the isolates were explored using the unsupervised machine learning methods k-mode and hierarchical clustering. This were done because diversity and information are required for the machine learning models to find patterns to classify the data by source. The models were trained on isolates from the sources meat processing factories, salmon processing factories, rural and urban environments, dairy farms, and slugs. Further, a couple of the models for each machine learning method were used to predict the sources for a set of isolates from clinical human cases of listeriosis without known sources.

Based on research on the different machine learning methods and earlier studies using machine learning for source attribution with similar input data (Tanui et al., 2022; Wright & König, 2019) the Random Forest method was anticipated to have the best performance among the tested methods. In this study the method proved to have some of the highest and the most stabile performances for the models across the different number of features in the training data sets and with two different MLST methods used to represent the data. It showed no signs of performance decline or enhancement due to the feature-selection. It can also be argued that it was among the easiest methods to use, as Random Forest models requires few hyperparameters to be tuned and a minimal amount of preprocessing of the data is required for the model to handle the data (Raschka & Mirjalili, 2019, pp. 100-103; Wright & König, 2019). The Support Vector Machine method did manage to get higher test performance than the Random Forest method for a few of the models, but overall had a more unstable performance for the models with different training data sets. The Support Vector Machine method was affected by the scale of the input data. By adding scaling for the Support Vector Machine models they often had a performance close to or even better than the Random Forest

models but the models without scaling generally had poor performance. The input data was categorical, and the values only represented category labels not numeric values, and by performing scaling the label values would wrongly be treated as numeric values. The last method that was implemented was a neural network which had a stable performance for most of the models with different training data sets but had an overall lower performance than both the Random Forest models and the Support Vector Machine models with scaling. The neural network was also the hardest machine learning method to implement and had high cost, both in time and computational resources. All the models implemented in this study were trained on isolates with different sources, and the two food associated sources, meat and salmon processing factories comprised 69% of the data set. All the models for the different machine learning methods generally had the highest predicting performance for the isolates from the food-associated sources. This indicated that machine learning might find patterns in allelic profiles that could be used to classify at least the isolates with food-associated sources. Listeriosis in humans is mainly a foodborne disease (Degré et al., 2010, pp. 223-226), this means the models were getting highest performance for the most likely sources of listeriosis in humans. Both the cgMLST and wgMLST data sets showed diversity among the isolates, and the clustering done by the unsupervised machine learning methods displayed some grouping of isolates with food-associated sources.

The data set in this study was not gathered with this study in mind and had as mentioned a very skewed distribution of isolate per source as seen in Table 3, and this has likely affected the results. There are only 24 isolates with slugs as source as opposed to 456 isolates with meat processing factories as source. Taking measures to balance the classes might have given the models higher predictive performance, and better performance for some of the not food-associated sources. Measures to balance the data set could make the data set smaller by down-sampling the largest classes giving the machine learning models less data to train on, which might lead to poorer generalisation and performance. Measures to balance the data set by up-sampling the smallest classes could introduce a lot of duplicates or be hard to implement because the features might have dependencies between each other because they are genes. For machine learning models imbalanced data sets can pose a problem as the models will be biased towards predicting the largest class, because this often will give high accuracy (Raschka & Mirjalili, 2019, pp. 211-221). There were no attempts to balance the classes in this study and instead a performance metric suitable for imbalanced data sets was chosen to

take the imbalance into account. The different F1-scores factor in both the recall and the precision of the model (Raschka & Mirjalili, 2019, pp. 211-221). Different F1-scores were therefore deemed better for evaluating the performance for the models, because the F1-scores are not only affected by correct predictions, but also incorrect predictions. The weighted F1-score was the primary metric for model performance as it additionally weights in the imbalance of the data set.

The preprocessing needed for the data was kept simple by using allelic profiles made with MLST methods directly as features for the machine learning models. Allelic profiles were made with both cgMLST and wgMLST. Several cgMLST schemas are already available for public use, making it easy to generate cgMLST profiles for isolates for future predictions and using the same schemas to make allelic profiles in different laboratories (Jolley et al., 2018; Moura et al., 2016). Adding additional accessory genes found using wgMLST could enhance the resolution and give better predictions if any of the accessory genes had a strong association to the source of isolates. The research done by Liao et al. (2023) indicated that some association can be found between certain accessory genes and source, as well as for some of the core genes. The wgMLST schema was made internally. The schema evaluation of the wgMLST schema is shown in Figure 17, and this assessment of the alleles that had been included in the schema did not report any problematic alleles. By setting thresholds to filter out loci with a lot of missing alleles, loci with little genetic insight for the data set in this study could be filtered out. Missing the allele of a specific locus could also give information, so loci with number of missing alleles below the threshold got their missing alleles encoded as "-1" to keep them. This was done in a similar way by Arning et al. (2021) when allelic profiles were used for source attribution of *Campylobacter jejuni*. After data cleaning the wgMLST data set had 762 loci more than the cgMLST data set, and these were then assumed to be the loci of accessory genes. Using allelic profiles directly introduced numbers representing the categories for the different allele types in the loci. As already mentioned, this was not assumed to be a large issue for the Random Forest method but could be problematic for both the Support Vector Machine and the neural network. Additional encoding steps to one-hot encode the features were considered but the high cardinality of the features made this hard to implement without increasing the dimensionality a lot for the two data sets. Making encoding or preprocessing steps to get numeric data instead of categorical data might have given better predicting abilities for some of the models, however other similar studies have

had success with using allelic profiles directly earlier (Arning et al., 2021; Munck et al., 2020; Tanui et al., 2022). Finding the simplest way to make machine learning models for source attribution seemed like a good approach before adding what might be unnecessary preprocessing steps.

Because some of the machine learning methods are sensitive to the range of the numbers in the input data the cgMLST and the wgMLST data sets were explored by finding the highest numbers assigned to the allele types in the profiles and comparing it with the highest number of distinct alleles counted per locus. A difference between the two MLST schemas used when assigning numbers to distinct allele types became clear. The cgMLST data set had a lower count of distinct alleles per loci but had some very high numbers assigned to some of the allele types compared to the wgMLST data set, as seen in Table 4. The external cgMLST schema had 371020 different alleles and they were found and labelled by analysis of a larger set of isolates than the data set in this study according to information given on the schema by the nomenclature server Chewie (Mamede et al., 2021). The internally made wgMLST schema had 67351 alleles according to the schema evaluation in Figure 17. So, it is only natural that some of the numbers assigned to the allele types in the cgMLST profiles would have higher values. It should be noted that the isolates with clinical cases as source were among the input genomes when the wgMLST schema was made and evaluated, but not during the data exploration. This causes a higher number of distinct alleles in the wgMLST schema than in the wgMLST data set during exploration, but not in the same extent as for the externally made cgMLST schema.

Further data exploration was done using Shannon entropy and unsupervised machine learning for clustering. The Shannon entropy expresses the diversity in the data sets and was used as a measure of information in the data sets. Having enough information in the data sets is important to make it possible for machine learning models to find patterns. The distribution of Shannon entropies for the loci in both the cgMLST and the wgMLST data sets can be seen in Figure 18, and the two data sets had similar distribution of entropies. Adding the accessory genes that the wgMLST data set contributed did not add any loci with notably higher entropy than the cgMLST data set, and the average entropy hardly changed. Some loci with no entropy were observed, and these had to have the same allele type value for all the isolates.

However, both data sets had numerous loci with diversity. Unsupervised machine learning methods using k-modes and hierarchical clustering were used to look for groupings and similarities between the isolates in the data sets. The results of the clustering are seen in Figure 19-22. As is apparent, the results were quite similar for both MLST data sets. Both clustering methods found a cluster with mostly isolates with meat processing factories as source, and some smaller grouping of isolates with mostly salmon processing factories as source. This means the allelic profiles for the isolates in those clusters were quite similar. The metadata for the isolates reveals that the isolates from food processing factories are part of the factories surveillance of bacteria. Quite few of the isolates are sampled from the same factory which can make them closely related. This might be part of the reason for higher similarities found for some groups of food-associated isolates. The findings by both the diversity measuring and the unsupervised machine learning methods indicate the data sets have captured information on the isolates, and this could be enough for machine learning models to classify the isolates by source.

Both data sets were divided into training and test data sets before feature-selection to avoid leaking information from the test data set to the model training. For feature-selection mutual information was calculated for features in the cgMLST training data set with the class variable holding the source labels giving each feature a mutual information score. The wgMLST data set does presumably contain the same loci used as features in the cgMLST data set, in addition to loci for accessory genes. Therefore, feature-selection was not performed on the wgMLST data set. This can of course be a false assumption for some of the loci as two different MLST schemas were used to define loci. As seen in Figure 23 several of the features in the cgMLST training data set has mutual information with the class variable holding the source labels. Features having mutual information with the class variable have a dependency with the class variable, meaning that knowing the value of the feature gives some information on the value of the class variable. A few features had no mutual information score, this is not surprising because some of the loci used as features had no entropy either. The mutual information scores were used to make different feature selected training data sets, all with features having mutual information scores above the average for the features in the data set as seen in Table 5. The loci with the 10% highest Shannon entropies in the cgMLST data set were compared with the loci used as features in the feature selected cgMLST training data set with 10% of the features. As stated in results section 3.3.1 a large amount of the loci with the

highest Shannon entropy were also found among the features with the highest mutual information scores. This visualises the need for diversity to be able to use genomic data to give information on the sources. The drawback of using methods to only measure dependencies between two variables for feature-selection is that the information held by a combination of features with the class variable is not calculated at all. Features might not have information on all the sources, but a combination of features might have much more information. Using mutual information for feature-selection might have deprived the machine learning models of the opportunity to find some of these combinations in the feature selected training data sets.

Using a grid search to find optimal hyperparameters for models during training can be time and resource consuming depending on how many hyperparameters are tuned. The optimal hyperparameters that were found can be seen in Table 6. The Random Forest method had a low time consumption, up to 6 minutes, for training the model and only the hyperparameter for number of decision trees was chosen to be tuned. The number of decision trees found as optimal is hard to interpret because of the stochasticity of the Random Forest method, and when different seeds were tested the model performance did not change but the optimal number of decision trees did. Compared with the Random Forest method the Support Vector Machine method both with and without scaling had a similar time consumption for training. The two hyperparameters cost for misclassification and gamma to scale the kernel function were chosen to be tuned. Different kernel functions could also have been tested, but to limit this study only the rbf-kernel was used. When comparing the Support Vector Machine method with and without scaling, the models without scaling chooses lower values for the hyperparameters than the one with scaling. It looks like the models without scaling in this study requires softer boundaries than the models which uses scaling. Softer boundaries meant it allowed more misclassification (Raschka & Mirjalili, 2019, pp. 79-90). The neural network had the most hyperparameters that can be tuned, but to lower the time and resources used to make a neural network only three were tuned. The number of nodes in the hidden layer, the dropout rate for the dropout layer and the learning rate of the optimiser were tuned. Even when narrowing down the options for tuning hyperparameters, and not repeating the cross validation in the grid search each neural network used around an hour to train. Each run of the grid search was not that time consuming with these restrictions, but the stochasticity of the models required the grid search to be run a few times to get an average training performance

to find the most optimal combination of hyperparameter values. Small learning rates, like the ones seen in Table 7 as optimal, for the optimiser increase the risk of the optimiser only leading the training of the model to a low, local loss value calculated by the loss function instead of finding a lower, global loss value (Raschka & Mirjalili, 2019, pp. 421-422). The neural network could possibly have gotten better performance by also tuning other hyperparameter like the batch size. The grid search used in this study explores all hyperparameter values given to it to find optimal combinations. The time and amount of resources the grid search used for the Random Forest and the Support Vector Machine methods were not that high, but this was not an equally good way of finding optimal hyperparameter for a neural network. Other less exhaustive searches for optimal hyperparameters could have made it easier to tune more hyperparameters without increasing the cost of time and resources too much for the neural network.

The training performances measured in weighted F1-scores, seen in Figure 24, were quite similar for each the different machine learning methods models with different training data sets, except for the Support Vector Machine models without scaling. Those models had lower training performance and more variety in the weighted F1-scores than the Support Vector Machine models with scaling, the Random Forest, and neural network models. The difference between training performance of the Support Vector Machine method with and without scaling reflects the sensitivity to the scale of the input data for the Support Vector Machine models in this study. The neural network method however seems less impacted by the scale of the input data than the Support Vector Machine method, and the models only has a little lower performance than the Random Forest models.

The training performance of the models gave a glimpse into the predicting power of the models, but to assess the generalisation and get a performance measure closer to the true performance, the models were evaluated by predicting the sources for the isolates in the test data set. Differences between the cgMLST data set and wgMLST data set were explored by making confusion matrices for the models trained on the two data sets. Only the models trained on the cgMLST training data set with all the features were used to compare against the models trained on the wgMLST training data set to not add the effect of feature-selection. Only the Support Vector Machine models without scaling had any notable change in

performance between the method's model trained on the cgMLST training data set with all the features and the model trained on the wgMLST training data set. Figure 25 shows the confusion matrices for the models trained on the cgMLST training data set with all features for the different machine learning methods, while Figure 26 shows the same for the models trained on the wgMLST training data set. The Support Vector Machine model without scaling seems to be impacted by the difference in the features scale caused by the difference in the assignment of numbers to allele types for the two MLST methods discussed earlier. The range of values per feature was smaller in the wgMLST data set. This seems to cause the Support Vector Machine model without scaling to go from bad predicting abilities to suddenly getting the same predicting abilities as the Support Vector Machine model with scaling when trained on the wgMLST training data set. The Figures 25/26 also shows a similar predicting ability for isolates from each source between the Random Forest, Support Vector Machine with scaling, and the neural network. Those models had good predicting abilities, with high rates of correctly classified isolates for the isolates from the food-associated sources, meat and salmon processing factories. The Random Forest and Support Vector Machine models with scaling had a bit better predicting ability for the isolates from the source rural and urban environments that the neural network. The isolates from rural and urban environments were most often misclassified as isolates with salmon as source. The isolates with dairy farms were only predicted correct about half of the time and often misclassified as isolates with rural and urban environments as source instead for the Random Forest, Support Vector Machine with scaling, and the neural network models. Isolates with the source slugs is clearly the hardest to predict correctly for all the models. The imbalance in the data set might affect the predicting ability for each source. The isolates from the largest sources in the data set were generally predicted correctly more often than isolates from the smaller sources. However, the largest sources, meat and salmon processing factory, were separated from each other quite well and generally had few isolates from other sources wrongly predicted as from them.

The F1-score calculated per source for the models in Figure 27 showed much the same as the confusion matrices but for all the models trained on the different training data sets. The isolates from the two food-associated sources were clearly easiest to classify correctly. The imbalance of the data set should also be kept in mind here when assessing the predicting abilities per source, as the models would have been biased towards classifying the largest sources. The not food-associated sources had much lower F1-scores with the lowest score for

the slugs with fewest isolates in the data set. It is a bit hard to conclude because of the class imbalance but it could also indicate that the not food-associated sources might not have equally restricted habitats for *L. monocytogenes* like the food-associated sources, making it harder to find general patterns for isolates from those sources. The study by Liao et al. (2023) found evidence that *Listeria* isolates from natural habitats such as soil had a range of factors that impacted adaptations of isolates found there, and the factors may vary between different places isolates are sampled from. The rural and urban environmental isolates in this study were sampled from a variety of different places and with different soil types, this can possibly give high diversity and less similarity within the isolates with environment as source.

The weighted F1-score, which weighs in the number of isolates per source, and the macro F1-score, which takes the average of the F1-score for each source, are seen for the different models in Figure 28. The different models in the Figure are the models trained on the different numbers of features for the cgMLST training data sets and on the wgMLST training data set. The weighted F1-scores weights in the larges sources as most important, here being the food-associated sources. The macro F1-score is an average of the F1-scores per source and the predictions of isolates from each source are equally important. The models have already shown some ability to predict isolates from the food-associated sources best, and this was also reflected in the F1-scores where the weighted F1-scores always were higher than the macro F1-score. The models with the Random Forest method showed high performance scores and the most stable scores across the different training data sets for the models. The highest performance for the Random Forest method was for the model trained on the data set with most features, the wgMLST training data set. For the Support Vector Machine with scaling and the neural network the performance was best for the models trained on around 30-40 percent of the features in the cgMLST training data set, and the performance looked like it was affected some by higher or lower number of features for the training data sets. A slight inconsistency in the otherwise apparent correlation between the weighted F1-score and the macro F1-score were observed for the test performance of the Support Vector Machine models with scaling trained on the feature selected cgMLST training data sets with 10 percent and 20 percent of the features. This happened because the two models did not predict any isolates with slugs as source. Because of the difference in calculating the F1-scores this affected the macro F1-score more than the weighted F1-score. The Support Vector Machine without scaling overall had the poorest performance, and the performance declined for the

models as the number of features increased for the cgMLST training data set for the model. Then the model performance increased again to get a similar score as the Support Vector Machine model with scaling when trained on the wgMLST training data set. Again, the difference in the assignment of numbers to allele types for the two MLST methods likely brought the features into a more similar scale for the wgMLST data set than the features in the cgMLST training data sets, making it easier for the model without scaling to classify the isolates correct. The lowest weighted F1-score recorded was 0.672, and it was for the Support Vector Machine model without scaling trained on the cgMLST training data set with all features. In comparison the best recorded weighted F1-score was 0.871, and it was for the Support Vector Machine model with scaling trained on the cgMLST training data set with 40 percent of the features. The methods Random Forest and Support Vector Machine with scaling had weighted F1-scores over 0.82 for all models. The neural network had mostly models with weighted F1-scores above 0.80. Similar to the performance result in the study by Tanui et al. (2022) and Arning et al. (2021) where different machine learning methods were compared for source attribution using allelic profiles as input data, the ensemble learners like Random Forest overall had the best performance also in this study. The Support Vector Machine models showed potential of high performance but were sensitive to the scale of the input data in this study. Making the performance of Support Vector Machine models between studies harder to compare because the scale of the input data affects the performance a lot. The neural network in this study was inspired by the shallow dense neural network in the study by Arning et al. (2021), and the trend seen by comparing the neural network performance with the performance of the Random Forest method in both studies comes out quite similar. The neural networks performed poorer than the Random Forest models, but not by that much. The exact performances might be a bit hard to compare because of differences in methods and that the attribution of source are for different bacteria species but the trends seen for the overall performance of the methods in this study is somewhat similar to what earlier studies in the same field has found.

The different machine learning methods were used to predict the sources for isolates from the clinical human cases. The chosen models were trained on the cgMLST training data set with all features and the wgMLST training data set. The predictions were only done to investigate the difference between the two methods of making MLST allelic profiles and not the effect of feature-selection. The models using the Random Forest, the Support Vector Machine with

scaling and the neural network methods showed most similarity in frequency of predicted sources. The Support Vector Machine model without scaling trained on the cgMLST training data set did only predict meat processing factory as source for all the clinical isolates, meaning it heavily preferred predicting the largest source it was trained with. This was similar to the predictions the model had on the test data set seen in the confusion matrix for the model in Figure 25, where most isolates were predicted to be from meat processing factories. The Support Vector Machine model without scaling trained on the wgMLST training data set did not predict any clinical isolates to have the source dairy farm, but all other sources were at least predicted ones. The infection source for the clinical cases were not known, but almost all models did predict slugs as source for at least some of the isolates. This is not very likely to be true but shows that machine learning models used for source attribution have the downside that they always will make a prediction and it is only possible to predict sources the model is trained on. This is similar to the limitations for the statistical model approaches to source attribution, like the models using microbial subtyping methods, that only can calculate probabilities for predefined sources (Pires et al., 2009).

5.0 Conclusion and future work

The methods used in this study shows that there are potential of using genomic data of *Listeria monocytogenes* sampled in Norway for source attribution using machine learning. Although the imbalance in the data set made it hard to conclude on the exact predicting abilities for the machine learning methods. The data set in this study was not that large but still had enough diversity to at least classify the food-associated sources using both cgMLST and wgMLST allelic profiles to represent the genomes. It looked like the effect of the feature-selection done in this study was small for the Random Forest method but had some effect for the Support Vector Machine and the neural network methods. Of the machine learning methods that were explored in this study the Random Forest method was the easiest to implement and overall got the best results. The method was also the one best suited for the categorical input data the MLST methods made. More research should be done to be able to make a good enough model to be used in the search of the source during a listeriosis outbreak.

For future work a data set with more balanced classes should be tested. Since the optimal goal is to make a model good at predicting sources for human clinical cases of listeriosis the sources should primary be food-associated. Measures to balance the data set in this study could also be attempted to see how that impacts the predictions compared to the imbalanced data set. If possible, a larger data set should also be tested to capture a larger part of the genetic diversity in the population. Better approaches to feature-selection could be investigated by annotating the loci in the MLST schema and selecting loci based on the genes associated with adaptations to the habitat like the ones found by Liao et al. (2023) for *L. monocytogenes*. The neural network has a large potential to be improved as only a few of the hyperparameters were tuned in this study. The tree-based ensemble learner Random Forest did well in this study, and other ensemble learners could be tested to see if better performance could be contained. Finally, a set of clinical isolates with known sources should be used to validate model performance.

References

Altschul, S. F., Gish, W., Miller, W., Meyers, E. W., & Lipman, D. J. (1990). Basic Local Alignment Search Tool. *Journal of molecular biology*, *215*(3), 403-410. https://doi.org/10.1006/jmbi.1990.9999

Arning, N., Sheppard, S. K., Bayliss, S., Clifton, D. A., & Wilson, D. J. (2021). Machine learning to predict the source of campylobacteriosis using whole genome data. *PLoS Genetics*, *17*(10). https://doi.org/10.1371/journal.pgen.1009436

Bayliss, S. C., Locke, R. K., Jenkins, C., Chattaway, M. A., Dallman, T. J., & Cowley, L. A. (2023). Rapid geographical source attribution of Salmonella enterica serovar Enteritidis genomes using hierarchical machine learning. *Elife*, *12*. https://doi.org/10.7554/eLife.84167

Castelli, P., De Ruvo, A., Bucciacchio, A., D'Alterio, N., Cammà, C., Di Pasquale, A., & Radomski, N. (2023). Harmonization of supervised machine learning practices for efficient source attribution of Listeria monocytogenes based on genomic data. *BMC genomics*, *24*(1). https://doi.org/10.1186/s12864-023-09667-w

Chambert-Loir, A. (2022). *Information Theory : Three Theorems by Claude Shannon* (1 ed., Vol. 144). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-031-21561-2

Degré, M., Hovig, B., & Rollag, H. (2010). *Medisinsk mikrobiologi* (3 ed.). Gyldendal.

Ditlefsen, A., & Egeland, E. S. (2023). *listeria*. Store norske leksikon. Retrieved 9. april 2024 from https://snl.no/listeria

Fagerlund, A., Idland, L., Heir, E., Møretrø, T., Aspholm, M., Lindbäck, T., & Langsrud, S. (2022). Whole-Genome Sequencing Analysis of Listeria monocytogenes from Rural, Urban, and Farm Environments in Norway: Genetic Diversity, Persistence, and Relation to Clinical and Food Isolates. *Applied and Environmental Microbiolgy*, *88*(6). https://doi.org/10.1128/aem.02136-21

Folkehelseinstituttet. (2023, 19. januar). *Smittevernveileder*. https://www.fhi.no/sm/smittevernveilederen/sykdommer-a-a/listeriose---veileder-for-helsepers/?term=

Graves, L. M., & Swaminathan, B. (2001). PulseNet standardized protocol for subtyping Listeria monocytogenes by macrorestriction and pulsed-field gel electrophoresis. *International journal of food microbiology*, *65*(1), 55-62. https://doi.org/10.1016/S0168-1605(00)00501-8

Hyatt, D., Chen, G.-L., Locascio, P. F., Land, M. L., Larimer, F. W., & Hauser, L. J. (2010). Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics*, *11*(1). https://doi.org/10.1186/1471-2105-11-119

Jolley, K. A., Bray, J. E., & Maiden, M. C. J. (2018). Open-access bacterial population genomics: BIGSdb software, the PubMLST.org website and their applications. *Wellcome Open Res*, *3*, 124-124. https://doi.org/10.12688/wellcomeopenres.14826.1

Kapperud, G. (2018, 17. desember). *Utbrudsveileder*. Folkehelseinstituttet. https://www.fhi.no/ut/utbruddsveilederen/?term=

Kaspersen, H., & Fiskebeck, E. Z. (2022). ALPPACA - A tooL for Prokaryotic Phylogeny And Clustering Analysis. *Journal of open source software*, *7*(79), 4677. https://doi.org/10.21105/joss.04677

Klug, W. S., Cummings, M. R., Spencer, C. A., & Palladino, M. A. (2013). *Essentials of genetics* (8th ed.). Pearson.

Liao, J., Guo, X., Li, S., Anupoju, S. M. B., Cheng, R. A., Weller, D. L., Sullivan, G., Zhang, H., Deng, X., & Wiedmann, M. (2023). Comparative genomics unveils extensive genomic variation between populations of Listeria species in natural and food-associated environments. *ISME Communications*, *3*(1). https://doi.org/10.1038/s43705-023-00293-x

Linke, K., Rückerl, I., Brugger, K., Karpiskova, R., Walland, J., Muri-Klinger, S., Tichy, A., Wagner, M., & Stessl, B. (2014). Reservoirs of listeria species in three environmental ecosystems. *Applied Environmental Microbiology*, *80*(18). https://doi.org/10.1128/AEM.01018-14

Lupolova, N., Lycett, S. J., & Gally, D. L. (2019). A guide to machine learning for bacterial host attribution using genome sequence data. *Microbial Genomics*, *5*(12). https://doi.org/10.1099/mgen.0.000317

Lyngstad, T. M., Lange , H., Brandal, L. T., Astrup, E., Eide, H. N., Johansen, T. B., Lund, H., Naseer, U., Amato, E., Grenersen, M. P., Lavoll, S. B., Jore, S., Soleng, A., Steinert, M., Grøneng, G. M., Salamanca, B. V., MacDonald, E., Nygård, K., & Feruglio, S. L. (2022). *Årsrapport 2021: Overvåkning av sykdommer som smitter fra mat, vann og dyr, inkludert vektorbårne sykdommer* [Yearly report]. https://www.fhi.no/globalassets/dokumenterfiler/rapporter/2022/2021_arsrapp_mat_vann_dyr.pdf

Maiden, M. C. J., Jansen van Rensburg, M. J., Bray, J. E., Earle, S. G., Ford, S. A., Jolley, K. A., & McCarthy, N. D. (2013). MLST revisited: the gene-by-gene approach to bacterial genomics. *Nature reviews Microbiology*, *11*(10), 728-736. https://doi.org/10.1038/nrmicro3093

Mamede, R., Vila-Cerqueira, P., Silva, M., Carriço, J. A., & Ramirez, M. (2021). Chewie Nomenclature Server (chewie-NS): a deployable nomenclature server for easy sharing of core and whole genome MLST schemas. *Nucleic Acids Research*, *49*(D1). https://doi.org/10.1093/nar/gkaa889

Martin, C. J. M., Bygraves, J. A., Feil, E., Morelli, G., Russell, J. E., Urwin, R., Zhang, Q., Zhou, J., Zurth, K., Caugant, D. A., Feavers, I. M., Achtman, M., & Spratt, B. G. (1998). Multilocus Sequence Typing: A Portable Approach to the Identification of Clones within Populations of Pathogenic Microorganisms. *Proceedings of the National Academy of Science of the U S A*, *95*(6), 3140-3145. https://doi.org/10.1073/pnas.95.6.3140

Maury, M. M., Tsai, Y.-H., Charlier, C., Touchon, M., Chenal-Francisque, V., Leclercq, A., Criscuolo, A., Gaultier, C., Roussel, S., Brisabois, A., Disson, O., Rocha, E. P. C., Brisse, S., & Lecuit, M. (2016). Uncovering Listeria monocytogenes hypervirulence by harnessing its biodiversity. *Nature Genetics*, *48*(3), 308-313. https://doi.org/10.1038/ng.3501

Moura, A., Criscuolo, A., Pouseele, H., Maury, M. M., Leclercq, A., Tarr, C., Björkman, J. T., Dallman, T., Reimer, A., Enouf, V., Larsonneur, E., Carleton, H., Bracq-Dieye, H., Katz, L. S., Jones, L., Touchon, M., Tourdjman, M., Walker, M., Stroika, S., . . . Brisse, S. (2016). Whole genome-based population biology and epidemiological surveillance of Listeria monocytogenes. *Nature Microbiologi*, *2*(2). https://doi.org/10.1038/nmicrobiol.2016.185

Mughini-Gras, L., Kooh, P., Fravalo, P., Augustin, J.-C., Guillier, L., David, J., Thébault, A., Carlin, F., Leclercq, A., Jourdan-Da-Silva, N., Pavio, N., Villena, I., Sanaa, M., & Watier, L. (2019). Critical Orientation in the Jungle of Currently Available Methods and Types of Data for Source Attribution of Foodborne Diseases. *Frontiers in Microbiology*, *10*. https://doi.org/10.3389/fmicb.2019.02578

Munck, N., Njage, P. M. K., Leekitcharoenphon, P., Litrup, E., & Hald, T. (2020). Application of Whole-Genome Sequences and Machine Learning in Source Attribution of Salmonella Typhimurium. *Risk Analysis*, *40*(9), 1693-1705. https://doi.org/10.1111/risa.13510

Nemoy, L. L., Kotetishvili, M., Tigno, J., Keefer-Norris, A., Harris, A. D., Perencevich, E. N., Johnson, J. A., Torpey, D., Sulakvelidze, A., Morris, J. G., & Stine, O. C. (2005). Multilocus Sequence Typing versus Pulsed-Field Gel Electrophoresis for Characterization of Extended-Spectrum Beta-Lactamase-Producing Escherichia coli Isolates. *Journal of Clinical Microbiology*, *43*(4), 1776-1781. https://doi.org/10.1128/JCM.43.4.1776-1781.2005

NicAogáin, K., & O'Byrne, C. P. (2016). The Role of Stress and Stress Adaptations in Determining the Fate of the Bacterial Pathogen Listeria monocytogenes in the Food Chain. *Frontiers Microbiology*, *7*. https://doi.org/10.3389/fmicb.2016.01865

Painset, A., Björkman, J. T., Kiil, K., Guillier, L., Mariet, J.-F., Félix, B., Amar, C., Rotariu, O., Roussel, S., Perez-Reche, F., Brisse, S., Moura, A., Lecuit, M., Forbes, K., Strachan, N., Grant, K., Møller-Nielsen, E., & Dallman, T. J. (2019). LiSEQ - whole-genome sequencing of a cross-sectional survey of Listeria monocytogenes in ready-to-eat foods and human clinical cases in Europe. *Microbial Genomomics*, *5*(2). https://doi.org/10.1099/mgen.0.000257

Pires, S. M., Evers, E. G., Pelt, W. v., Ayers, T., Scallan, E., Angulo, F. J., Havelaar, A., & Hald, T. (2009). Attributing the Human Disease Burden of Foodborne Infections to Specific Sources. *Foodborne Pathog and Disease*, *6*(4), 417-424. https://doi.org/10.1089/fpd.2008.0208

Raschka, S., & Mirjalili, V. (2019). *Python machine learning: machine learning and deep learning with Python, scikit-learn, and TensorFlow* (Third edition ed.). Packt Publishing.

Rasko, D. A., Myers, G. S. A., & Ravel, J. (2005). Visualization of comparative genomic analyses by BLAST score ratio. *BMC Bioinformatics*, *6*(1). https://doi.org/10.1186/1471-2105-6-2

Sherwin, W. B. (2010). Entropy and Information Approaches to Genetic Diversity and its Expression: Genomic Geography. *Entropy*, *12*(7), 1765-1798. https://doi.org/10.3390/e12071765

Silva, M., Machado, M. P., Silva, D. N., Rossi, M., Moran-Gilad, J., Santos, S., Ramirez, M., & Carriço, J. A. (2018). chewBBACA: A complete suite for gene-by-gene schema creation and strain identification. *Microb Genom*, *4*(3). https://doi.org/10.1099/mgen.0.000166

Tanui, C. K., Benefo, E. O., Karanth, S., & Pradhan, A. K. (2022). A Machine Learning Model for Food Source Attribution of Listeria monocytogenes. *Pathogens*, *11*(6). https://doi.org/10.3390/pathogens11060691

Tortora, G. J., Case, C. L., & Funke, B. R. (2004). *Microbiology : an introduction* (8th , international ed.). Benjamin Cummings.

Wright, M. N., & König, I. R. (2019). Splitting on categorical predictors in random forests. *PeerJ*, *7*. https://doi.org/10.7717/peerj.6339

Appendix A

Test performance of models when predicting sources for isolates in test data set given in the metrics F1-score per source. Because of the 30 times each of the shallow dense neural network models predicted the sources for the isolates in the test data set the metrics are an average and comes with a standard deviation (SD).

F1-score dairy farm	F1-score meat processing factory	F1-score rural/urban environment	F1-score salmon processing factory	F1-score slugs	Training data set	Machine Learning method
0.5098039215688627	0.9283018867924453	0.5970149253731343	0.9214659685863874	0.25	cgMLST 10%	Random Forest
0.5238095238095524	0.9230769230769231	0.6666666666666667	0.9119170984455596	0.25	cgMLST 20%	Random Forest
0.5789473684210053	0.9197080291970802	0.6756756756756757	0.9304812834224599	0.2222222222222222	cgMLST 30%	Random Forest
0.5641025641025641	0.9230769230769231	0.6571428571428571	0.9214659685863874	0.2222222222222222	cgMLST 40%	Random Forest
0.5	0.9293680297397777	0.6857142857142857	0.9157894736842105	0.2222222222222222	cgMLST 50%	Random Forest
0.5652173913043478	0.9353612167300381	0.6760563380281690	0.9175257731958762	0.25	cgMLST all	Random Forest
0.5777777777777778	0.9477611940298511	0.7246376811594202	0.9270833333333333	0.25	wgMLST	Random Forest
0.4705882352941176	0.9241877256317691	0.5396825396825397	0.9398907103825137	0	cgMLST 10%	Support Vector Machine (scaling)
0.5416666666666667	0.9044117647058824	0.6086956521739130	0.9139784946236559	0	cgMLST 20%	Support Vector Machine (scaling)
0.5490196078431373	0.9056603773584906	0.6	0.9042553191489362	0.25	cgMLST 30%	Support Vector Machine (scaling)
0.56	0.9338235294117647	0.7272727272727273	0.96	0.25	cgMLST 40%	Support Vector Machine (scaling)
0.6341463414634146	0.9179104477611940	0.6117647058823529	0.9555555555555556	0.25	cgMLST 50%	Support Vector Machine (scaling)
0.6153846153846154	0.9363295880149812	0.6	0.9325842696629213	0.25	cgMLST all	Support Vector Machine (scaling)
0.5882352941176471	0.9318181818181818	0.5977011494252873	0.9206349206349206	0.25	wgMLST	Support Vector Machine (scaling)
0.4242424242424242	0.8490566037735849	0.4680851063829787	0.9431818181818182	0.25	cgMLST 10%	Support Vector Machine (without scaling)

0.5	0.820359281437126	0.3902439024390244	0.8982035928143712	0.25	cgMLST 20%	Support Vector Machine (without scaling)
0.5	0.810650887573964	0.35	0.9024390243902440	0.25	cgMLST 30%	Support Vector Machine (without scaling)
0.5	0.798833819241983	0.35	0.8679245283018870	0.25	cgMLST 40%	Support Vector Machine (without scaling)
0.5	0.794202898550725	0.3	0.8535031847133760	0.25	cgMLST 50%	Support Vector Machine (without scaling)
0.4	0.767507002801120	0.2564102564102560	0.7837837837837840	0.25	cgMLST all	Support Vector Machine (without scaling)
0.5	0.950943396226415	0.625	0.9392265193370170	0.25	wgMLST	Support Vector Machine (without scaling)
0.4278059528902160 SD: 0.08848844	0.8822660898465595 SD: 0.01070012	0.5184942688371580 SD: 0.05787652	0.9005636436104800 SD: 0.02449498	0.138577348577349 SD:0.1045266	cgMLST 10%	Shallow dense neural network
0.4978041129300900 SD: 0.06187045	0.9074897266996290 SD: 0.01159470	0.6228935252070630 SD: 0.05566155	0.9067755279969700 SD: 0.02278989	0.147829762829763 SD:0.1392345	cgMLST 20%	Shallow dense neural network
0.5572020887292600 SD: 0.06912250	0.9109511099601070 SD: 0.01343732	0.6147241982608280 SD: 0.04872228	0.9124608349296380 SD: 0.02417597	0.200268250268250 SD:0.1089546	cgMLST 30%	Shallow dense neural network
0.5445325434454449 SD: 0.09150846	0.9148228098919320 SD: 0.01542516	0.6125141519427030 SD: 0.05822710	0.9066466703657920 SD: 0.01865160	0.200287490287490 SD:0.1191153	cgMLST 40%	Shallow dense neural network
0.5330286196127410 SD: 0.06707779	0.9141733156781320 SD: 0.01642146	0.6101004198224250 SD: 0.06172945	0.9039307208692 SD: 0.02593997	0.187302419802420 SD:0.1332037	cgMLST 50%	Shallow dense neural network
0.5038151451567860 SD: 0.06710917	0.9066275162966100 SD: 0.01532435	0.6012246972601950 SD: 0.07445505	0.8960457961609170 SD: 0.01831366	0.157359677359677 SD:0.1383040	cgMLST all	Shallow dense neural network
0.4716023102989280 SD: 0.09738710	0.8648091017138280 SD:0.03293494	0.6004663937345600 SD: 0.04789428	0.8484284403213400 SD: 0.04664354	0.055555555555556 SD:0.1026470	wgMLST	Shallow dense neural network

Appendix B

Test performance of models when predicting sources for isolates in test data set given in the metrics weighted F1-score and macro F1-scores. Because of the 30 times each of the shallow dense neural network models predicted the sources for the isolates in the test data set the metrics are an average and comes with a standard deviation (SD).

Weighted F1	Macro F1	Training data set	Machine Learning method
0.837787224461517	0.64131734046412	cgMLST 10%	Random Forest
0.841427855302648	0.655094042399742	cgMLST 20%	Random Forest
0.850484210319517	0.665406915787698	cgMLST 30%	Random Forest
0.845956037848951	0.657602107026191	cgMLST 40%	Random Forest
0.845115462852424	0.650618802272099	cgMLST 50%	Random Forest
0.853425714178275	0.668832143851686	cgMLST all	Random Forest
0.868764583254354	0.685451997260077	wgMLST	Random Forest
0.825799050867908	0.574869842198188	cgMLST 10%	Support Vector Machine (scaling)
0.82216299932156	0.593750515634024	cgMLST 20%	Support Vector Machine (scaling)
0.825377735428704	0.641787060870113	cgMLST 30%	Support Vector Machine (scaling)
0.871215888417223	0.686219251336898	cgMLST 40%	Support Vector Machine (scaling)
0.855365906640973	0.673875410132504	cgMLST 50%	Support Vector Machine (scaling)
0.854051438477478	0.666859694612504	cgMLST all	Support Vector Machine (scaling)
0.84573202321681	0.657677909199207	wgMLST	Support Vector Machine (scaling)
0.785518024212375	0.586913190516161	cgMLST 10%	Support Vector Machine (without scaling)
0.755517504091641	0.571761355338104	cgMLST 20%	Support Vector Machine (without scaling)
0.747693071452767	0.562617982392842	cgMLST 30%	Support Vector Machine (without scaling)
0.73145512296674	0.553351669508774	cgMLST 40%	Support Vector Machine (without scaling)
0.719144617613928	0.53954121665282	cgMLST 50%	Support Vector Machine (without scaling)
0.67182315596506	0.491540208599032	cgMLST all	Support Vector Machine (without scaling)
0.856304577399829	0.653033983112686	wgMLST	Support Vector Machine (without scaling)
0.791301984243847 SD: 0.01708532	0.573541460752359 SD: 0.03233507	cgMLST 10%	Shallow dense neural network
0.822932932850653 SD: 0.01533288	0.616558531132703 SD: 0.03831344	cgMLST 20%	Shallow dense neural network

0.831554651657008 SD: 0.01695529	0.639121296429617 SD: 0.03253973	cgMLST 30%	Shallow dense neural network
0.830284143563328 SD: 0.02051635	0.635760733186673 SD: 0.04210286	cgMLST 40%	Shallow dense neural network
0.82760352848658 SD: 0.01929962	0.629707099156984 SD: 0.03148468	cgMLST 50%	Shallow dense neural network
0.81747624612366 SD: 0.01819976	0.613014566446837 SD: 0.03733560	cgMLST all	Shallow dense neural network
0.777869903412436 SD: 0.03540770	0.559866637023205 SD: 0.03799800	wgMLST	Shallow dense neural network

Appendix C

The predictions of the sources for the isolates from clinical human cases made by the models trained on the cgMLST training data set with all features and the wgMLST training data set.

SRA_no	Random Forest cgMLST	Random Forest wgMLST	Support Vector Machine (scaling) cgMLST	Support Vector Machine (scaling) wgMLST	Support Vector Machine (without scaling) cgMLST	Support Vector Machine (without scaling) wgMLST	Shallow dense neural network cgMLST	Shallow dense neural network wgMLST
ERR2522241	salmon	salmon	rural/urban	rural/urban	meat	rural/urban	dairy farm	salmon
ERR2522242	salmon	salmon	rural/urban	salmon	meat	salmon	salmon	salmon
ERR2522243	salmon	salmon	meat	salmon	meat	salmon	salmon	meat
ERR2522244	salmon	meat	rural/urban	rural/urban	meat	rural/urban	rural/urban	meat
ERR2522245	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522246	salmon	salmon	salmon	salmon	meat	rural/urban	salmon	salmon
ERR2522247	meat	meat	meat	meat	meat	meat	meat	meat
ERR2522248	meat	meat	meat	meat	meat	meat	meat	meat
ERR2522249	rural/urban	rural/urban	meat	rural/urban	meat	rural/urban	salmon	slugs
ERR2522250	rural/urban	rural/urban	meat	rural/urban	meat	rural/urban	meat	slugs
ERR2522251	rural/urban	salmon	rural/urban	rural/urban	meat	rural/urban	meat	salmon
ERR2522252	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522253	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	salmon	meat
ERR2522254	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522255	rural/urban	rural/urban	meat	rural/urban	meat	rural/urban	salmon	slugs
ERR2522256	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522257	meat	meat	meat	meat	meat	meat	salmon	meat
ERR2522258	salmon	salmon	salmon	salmon	meat	rural/urban	salmon	salmon
ERR2522259	dairy farm	dairy farm	rural/urban	salmon	meat	rural/urban	salmon	rural/urban
ERR2522260	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522261	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522262	meat	meat	meat	meat	meat	meat	meat	meat
ERR2522263	salmon	salmon	salmon	salmon	meat	meat	meat	salmon
ERR2522264	meat	meat	rural/urban	rural/urban	meat	rural/urban	slugs	salmon
ERR2522265	meat	meat	dairy farm	meat	meat	rural/urban	meat	meat
ERR2522266	meat	meat	meat	meat	meat	rural/urban	meat	meat

ERR2522267	salmon	salmon	rural/urban	rural/urban	meat	rural/urban	salmon	salmon
ERR2522268	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522269	meat	meat	meat	meat	meat	meat	meat	meat
ERR2522270	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522271	salmon	salmon	salmon	meat	meat	rural/urban	salmon	salmon
ERR2522272	dairy farm	dairy farm	rural/urban	rural/urban	meat	rural/urban	rural/urban	dairy farm
ERR2522273	meat	meat	meat	rural/urban	meat	rural/urban	meat	meat
ERR2522274	slugs	slugs	slugs	slugs	meat	slugs	meat	slugs
ERR2522275	salmon	salmon	salmon	salmon	meat	rural/urban	salmon	meat
ERR2522276	salmon	salmon	salmon	salmon	meat	meat	salmon	salmon
ERR2522277	meat	meat	meat	meat	meat	meat	salmon	salmon
ERR2522278	salmon	salmon	salmon	salmon	meat	salmon	salmon	meat
ERR2522279	salmon	salmon	salmon	salmon	meat	rural/urban	salmon	salmon
ERR2522280	salmon	salmon	salmon	meat	meat	rural/urban	salmon	salmon
ERR2522281	salmon	meat	rural/urban	rural/urban	meat	rural/urban	rural/urban	meat
ERR2522282	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	salmon	meat
ERR2522283	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522284	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522285	salmon	meat	rural/urban	rural/urban	meat	rural/urban	dairy farm	salmon
ERR2522286	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522287	meat	meat	meat	meat	meat	meat	dairy farm	meat
ERR2522288	meat	meat	meat	meat	meat	rural/urban	meat	meat
ERR2522289	meat	meat	meat	meat	meat	meat	meat	meat
ERR2522290	salmon	salmon	meat	rural/urban	meat	rural/urban	salmon	rural/urban
ERR2522291	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	salmon	dairy farm
ERR2522292	salmon	salmon	rural/urban	salmon	meat	salmon	salmon	salmon
ERR2522293	meat	meat	rural/urban	rural/urban	meat	rural/urban	salmon	salmon
ERR2522294	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522295	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522296	dairy farm	dairy farm	rural/urban	dairy farm	meat	rural/urban	salmon	dairy farm
ERR2522297	salmon	salmon	salmon	salmon	meat	meat	salmon	salmon
ERR2522298	salmon	rural/urban	salmon	rural/urban	meat	rural/urban	salmon	dairy farm
ERR2522299	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	salmon	meat

ERR2522300	salmon	meat	rural/urban	rural/urban	meat	rural/urban	rural/urban	meat
ERR2522301	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	salmon	rural/urban
ERR2522302	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	salmon	meat
ERR2522303	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	meat
ERR2522304	rural/urban	meat	rural/urban	rural/urban	meat	rural/urban	meat	meat
ERR2522305	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522306	salmon	salmon	salmon	rural/urban	meat	rural/urban	salmon	rural/urban
ERR2522307	dairy farm	dairy farm	rural/urban	dairy farm	meat	rural/urban	rural/urban	dairy farm
ERR2522308	salmon	salmon	rural/urban	rural/urban	meat	rural/urban	dairy farm	salmon
ERR2522309	meat	meat	meat	rural/urban	meat	rural/urban	salmon	rural/urban
ERR2522310	meat	meat	meat	rural/urban	meat	rural/urban	meat	meat
ERR2522311	meat	meat	meat	meat	meat	meat	meat	meat
ERR2522312	salmon	salmon	meat	salmon	meat	meat	salmon	salmon
ERR2522313	salmon	meat	rural/urban	rural/urban	meat	rural/urban	salmon	meat
ERR2522314	dairy farm	dairy farm	dairy farm	dairy farm	meat	meat	dairy farm	rural/urban
ERR2522315	rural/urban	rural/urban	rural/urban	salmon	meat	rural/urban	rural/urban	salmon
ERR2522316	meat	meat	meat	rural/urban	meat	rural/urban	salmon	salmon
ERR2522317	meat	meat	salmon	rural/urban	meat	rural/urban	salmon	rural/urban
ERR2522318	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522319	salmon	salmon	salmon	meat	meat	rural/urban	salmon	salmon
ERR2522320	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522321	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522322	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522323	dairy farm	dairy farm	dairy farm	dairy farm	meat	meat	dairy farm	dairy farm
ERR2522324	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522325	salmon	salmon	salmon	salmon	meat	salmon	meat	meat
ERR2522326	slugs	slugs	rural/urban	rural/urban	meat	rural/urban	salmon	salmon
ERR2522327	salmon	salmon	rural/urban	rural/urban	meat	rural/urban	dairy farm	salmon
ERR2522328	salmon	salmon	salmon	rural/urban	meat	rural/urban	meat	rural/urban
ERR2522329	salmon	salmon	salmon	rural/urban	meat	rural/urban	salmon	rural/urban
ERR2522330	slugs	slugs	salmon	rural/urban	meat	rural/urban	salmon	meat
ERR2522331	salmon	salmon	salmon	rural/urban	meat	rural/urban	meat	rural/urban
ERR2522332	dairy farm	dairy farm	rural/urban	salmon	meat	rural/urban	rural/urban	dairy farm

ERR2522333	dairy farm	dairy farm	meat	salmon	meat	rural/urban	meat	salmon
ERR2522334	salmon	salmon	meat	salmon	meat	salmon	salmon	meat
ERR2522335	salmon	salmon	salmon	salmon	meat	rural/urban	salmon	salmon
ERR2522336	meat	meat	salmon	rural/urban	meat	rural/urban	salmon	dairy farm
ERR2522337	salmon	salmon	salmon	salmon	meat	salmon	meat	salmon
ERR2522338	salmon	salmon	meat	salmon	meat	salmon	salmon	salmon
ERR2522339	rural/urban	rural/urban	salmon	rural/urban	meat	rural/urban	meat	rural/urban
ERR2522340	salmon	salmon	salmon	rural/urban	meat	rural/urban	salmon	meat
ERR2522341	dairy farm	dairy farm	dairy farm	rural/urban	meat	rural/urban	meat	dairy farm
ERR2522342	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522343	salmon	salmon	salmon	rural/urban	meat	rural/urban	salmon	rural/urban
ERR2522344	rural/urban	rural/urban	salmon	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522345	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522346	salmon	salmon	salmon	salmon	meat	salmon	meat	meat
ERR2522347	salmon	salmon	salmon	rural/urban	meat	rural/urban	salmon	rural/urban
ERR2522348	salmon	salmon	salmon	rural/urban	meat	rural/urban	rural/urban	salmon
ERR2522349	dairy farm	dairy farm	dairy farm	dairy farm	meat	rural/urban	salmon	dairy farm
ERR2522350	meat	meat	meat	rural/urban	meat	rural/urban	meat	meat
ERR2522351	salmon	salmon	salmon	rural/urban	meat	rural/urban	salmon	rural/urban
ERR2522352	salmon	salmon	salmon	rural/urban	meat	rural/urban	salmon	meat
ERR2522353	salmon	salmon	salmon	salmon	meat	salmon	salmon	salmon
ERR2522355	meat	salmon	rural/urban	rural/urban	meat	rural/urban	salmon	meat
ERR2522356	dairy farm	dairy farm	rural/urban	rural/urban	meat	rural/urban	meat	rural/urban
ERR2522357	dairy farm	dairy farm	rural/urban	rural/urban	meat	rural/urban	meat	rural/urban
ERR2522359	salmon	salmon	rural/urban	salmon	meat	salmon	salmon	salmon
ERR2522360	meat	meat	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522361	dairy farm	dairy farm	dairy farm	dairy farm	meat	rural/urban	salmon	dairy farm
ERR2522362	salmon	salmon	salmon	meat	meat	rural/urban	salmon	salmon
ERR2522363	salmon	salmon	salmon	salmon	meat	salmon	meat	salmon
ERR2522364	salmon	meat	rural/urban	rural/urban	meat	rural/urban	rural/urban	meat
ERR2522365	meat	meat	meat	meat	meat	meat	salmon	meat
ERR2522366	salmon	meat	rural/urban	rural/urban	meat	rural/urban	rural/urban	meat
ERR2522367	salmon	salmon	meat	rural/urban	meat	rural/urban	meat	meat

ERR2522368	rural/urban	rural/urban	rural/urban	rural/urban	meat	rural/urban	rural/urban	rural/urban
ERR2522369	slugs	slugs	slugs	slugs	meat	slugs	meat	slugs
ERR2522370	meat	meat	rural/urban	rural/urban	meat	rural/urban	slugs	salmon
ERR3047199	salmon	dairy farm	rural/urban	rural/urban	meat	rural/urban	rural/urban	salmon
ERR3446056	rural/urban	rural/urban	rural/urban	salmon	meat	rural/urban	dairy farm	salmon

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway