Norwegian University
of Life Sciences

Master's Thesis 2022 30 ECTS
Faculty of Chemistry Biotechnology and Food Sciences

DeepGene – Gene finding based on upstream sequence data

Trude Haug Almestrand
M- KB Bioinformatics

Abstract

Genome annotation is a process of identifying functional elements along a genome. By correctly locating and finding the information stored within a sequence, knowledge about structural features and functional roles can be revealed. With the number of sequences doubling approximately every 18 months, there is a severe need for automatic annotation of genomes. Today there are many different annotation software tools available, however they produce far from perfect results.

Here a new project, DeepGene, is presented. Using data from the RefSeq prokaryotic database we have started an effort to improve on the prokaryotic genome annotation process.
This thesis presents the initial efforts of said improvement with a focus on discerning between coding and non-coding sequences using upstream sequence data from open reading frames. Using the 15 prokaryotic genomes available in the RefSeq database, upstream data was retrieved and processed into two datasets, and were then trained using several popular classification models. The performance of the models was compared with a standard annotation tool to create a general baseline for our model. The models created from the datasets show many similarities in terms of metrics. With the K-mer data having a mean precision at 0.22 and mean recall of 0.74, and the sequential data having a mean precision at 0.30 and mean recall at 0.77. Both the datasets performed worse than our standard annotation software with a mean recall and precision of, respectively, 0.83 and 0.82. As far as upstream sequences are concerned, the models managed to pull all the information available from both datasets. The initial results gave limited information in terms of classification and motif presence indicating that other attributes surrounding the genome should be looked at for a possible improvement on the annotation problem. An ideal step forward is to expand into a pipeline so that the complex false negative classifications may be explained.

Sammendrag

Genomannotering er en prosess som skal identifisere funksjonelle elementer langs et genom. Ved å finne informasjonen lagret i en sekvens kan man avsløre kunnskap rundt strukturelle og funksjonelle roller. Ettersom antall sekvenser dobler rundt hver 18. måned er det et sterkt behov for automatisk gjenkjenning av genomer. I dag er det mange tilgjengelige annoteringsverktøy, men de produserer langt fra perfekte resultater.

Et nytt prosjekt ved navn DeepGene er her presentert. Ved hjelp av data fra RefSeq prokaryotiske database har vi startet et forsøk på å forbedre den prokaryotiske annoteringsprosessen. I denne oppgaven presenteres begynnelsen på forbedringen. Hovedfokuset var å skille mellom kodende og ikke-kodende sekvenser ved hjelp av sekvensdata oppstrøms for åpne leserammer. Ved å benytte seg av de 15 prokaryotiske genomene tilgjengelig i RefSeq databasen, ble oppstrømsdata hentet og prosessert til to datasett. Disse datasettene ble videre trent ved hjelp av populære klassifiseringsmodeller. Ytelsen til disse modellene ble sammenlignet med et standard annoteringsverktøy for å lage et generelt utgangspunkt til vår modell. Modellene trent av datasettet viser mange likheter når det kommer til ytelse. K-mer datasettet hadde en gjennomsnittlig presisjon på 0.22 og nøyaktighet på 0.74. Videre hadde det sekvensielle datasettet en gjennomsnittlig presisjon på 0.30 og en nøyaktighet på 0.77. Begge datasettene hadde dårligere resultater enn vårt standard annoteringsverktøy som hadde en gjennomsnittlig nøyaktighet og presisjon på henholdsvis 0.83 og 0.82. Når det kommer til oppstrømssekvenser klarer modellene å hente ut all informasjon tilgjengelig fra datasettene. Resultatene ga begrenset med informasjon når det kommer til klassifisering og motif-tilstedeværelse. Denne begrensningen indikerer at andre attributter rundt genomet bør undersøkes for en mulig forbedring rundt annoteringsproblemet. Et ideelt steg videre er å utvide modellene til en «pipeline» slik at komplekse falske negative klassifiseringer kan bli forklart.

Acknowledgements

The work presented in this thesis is the first part of a larger project dubbed DeepGene at the faculty of chemistry and biotechnology, NMBU.

I would like to thank my main supervisor Lars Snipen, and co-supervisor Kristian Liland for allowing me to be the groundbreaker in this exciting new project. I would also like to thank them for the many good discussions, as well as their great feedback and guidance.

Trude Haug Almestrand

Table of contents

List of figures

List of Abbreviations

CDS	Coding sequence
DNA	Deoxyribonucleic acid
GFF	General Feature Formats
KNN	K Nearest Neighbors
LDA	Linear Discriminant Analysis
LORF	Long open reading frame
nt	Nucleotides
ORF	Open Reading Frame
PC	Principal component
PGAP	Prokaryotic Genome Annotation Pipeline
PLS	Partial Least Square
PLS + LDA	Partial Least-Squares + Linear Discriminant Analysis
RBS	Ribosomal Binding Site
RNA	Ribonucleic acid
SD	Shine-Dalgarno

1. Introduction

1.1 Finding Genes

Prokaryotes are the most primitive and ancient form of life. They are also the most abundant and diverse empire on earth. No other organisms are as adaptable, and they are to be observed everywhere, even places humans cannot travel. From hydrothermal vents that can reach a temperature as high as 464 ℃ to frozen wastelands that can reach as low as -98,6 ℃. The current discovered temperature that microbial life can survive extends from -25 ℃ to 130 ℃ (Martinez-Cano et al., 2014), underlining the diversity present in the prokaryotic genome.

The Human Genome project marked a milestone within the biological world after its completion around 20 years ago (Lander et al., 2001). Not only has the project given enormous contributions to science, it has also started a revolutionary development in the world of bioinformatics. Today, sequencing is performed near continuously, and many genomes (finished and unfinished) are easily available.

Public databases are the medium in which genome sequences are published. The databases are important resources in biosciences. The public access allows researchers to utilize the data in research allowing greater innovation and information surrounding genomes. Newer sequencing technology has made it cheaper and more available to sequence the genome, making it even easier to generate new data. Figure 1.1.1 illustrates the cost per raw megabase of DNA sequence over the course of 20 years.

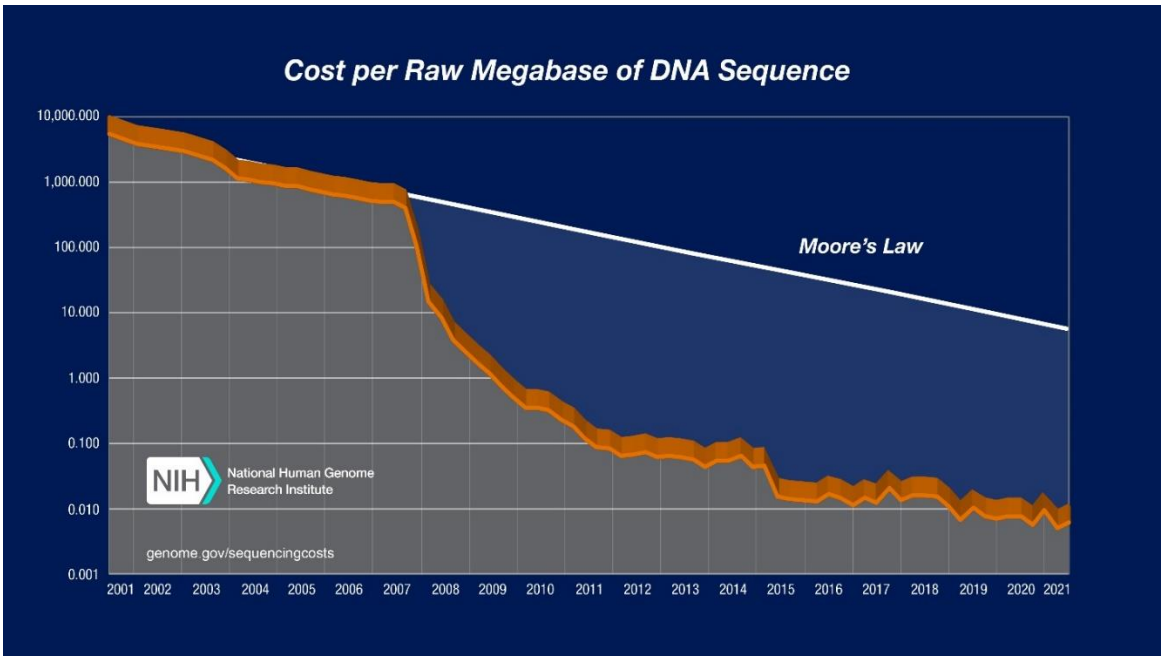


Figure 1.1.1 Sequencing cost per megabase retrieved from (NHGRI, 2022).

The amount of information does not seem to stagnate. Figure 1.1.2 illustrates the high amount of sequencing data available in GenBank, one of the largest DNA-sequence databases in the world. From 1982 to present data, the number of bases in GenBank has doubled approximately every 18 months (NCBI, 2022).

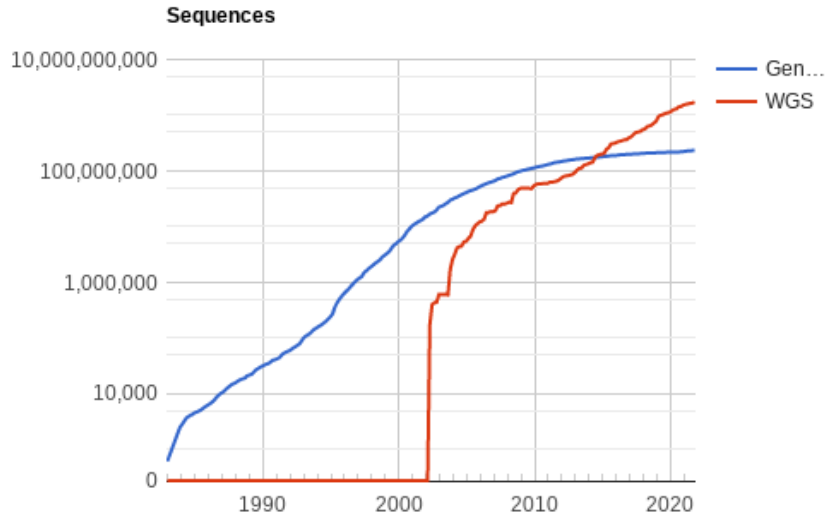


Figure 1.1.2 The growth of the GenBank database over the past decades. Red line shows the whole genome shotgun (WGS) projects, the blue line is GenBank sequences. Figure retrieved from (NCBI, 2022).

With the increase of high throughput sequencing, a bigger demand for automating the annotation of these sequences is required. Annotation of genomes is a multi-level process that involves prediction of protein-coding genes and other elements (Abril, 2019). A very basic, but essential part of annotation is to locate the start and end position of the coding genes.

There are currently many existing software tools that automates the annotation process. These tools mainly focus on homology-based methods to classify gene products (Armstrong et al., 2019; Galperin et al., 2019; Tatusova et al., 2016). An issue with these types of tools is their limitation in discovering novel genes (Anders et al., 2021).

The existing annotation tools also produce far from perfect results, with several both false positives and negatives (Dong et al., 2021). Some tools have self-stated error rates at around 4 – 6% per sequence (Lomsadze et al., 2018). When analyzing multiple sequences at a time this error rate can quickly increase, leading to many false annotations. With an error rate of 4%, a prokaryotic genome with 3000 genes gives 120 errors per genome. For a single genome, this may not seem as much, but if applied to many genomes, the databases can quickly be filled with false annotations. Especially when considering the already available sequences illustrated in figure 1.1.2.

The large part of the issue lies within the classification of correct start position for the genes. Because of overlapping Open Reading Frames some software have difficulties finding the correct frame (Palleja et al., 2008). Using modern methods from Data Science the DeepGene project has started an effort to improve on the prokaryotic genome annotation process.

1.2 Aim of study

The aim of this study is to investigate if sequences upstream of a start codon in an ORF is informative enough to discern between coding and non-coding ORFs, and in the long run locate the start position of a coding sequence in prokaryotic DNA.

This meant creating and processing a dataset based on the RefSeq prokaryotic genomes and retrieving upstream sequences from all genomes respective fasta file. The final process consisted of turning the sequence data into a numerical one for each genome and train a machine learning model.

In this thesis we will initially focus on finding the correct start position of a coding gene by utilizing existing theory from molecular biology. Firstly, we will see how well an existing prokaryotic genome annotation software can find the correct start codon.

Afterwards we will attempt to establish a set of *training data* i.e., a set of genomes where we have the most reliable information on where genes are found. We have started out with the 15 NCBI Reference genomes for prokaryotes. These being the best manual annotations we have as of today. Using the training data, we will use multiple modern classification machine learning models to see if there is any information to gather from the upstream sequence data. The information being whether it is possible to recognize the start of a gene based on their respective upstream sequence.

2. Background

2.1 The Prokaryotic Genome

In this subchapter certain molecular characteristics surrounding the prokaryotic genome will be addressed. Specifically, the more general existing theories surrounding upstream sequences will be described in detail, as well as some general characteristics surrounding prokaryotic DNA. The upstream sequences play an important part in the prokaryotic metabolic process, with many elements present a few bases upstream of a start codon.

The prokaryotic genome is primarily a circular, double stranded piece of DNA. DNA consist of four bases: Adenine, Cytosine, Guanine and Thymine, which together form the information of the genome. The backbone that supports the bases consists of every other phosphate and the sugar deoxyribose. The combination of the sugar molecule, the phosphate group and a nitrogen-containing base is called a nucleotide.

The blend of the four bases creates sequences, where some contain information. The sequences coding for information or Coding Sequence (CDS) are partitioned into three bases at a time. These three bases together are called a codon and codes for an amino acid. The length of genomes varies but is in general a few million base pairs long (Land et al., 2015). The information present in the sequences are so diverse, but many prokaryotes also share common CDS'. Small variations in the CDS allow them to thrive in different locations yet retain elements of the same machinery that gave them life.

Processing the information from the CDS to a polypeptide chain requires multiple steps. Firstly, the information from a CDS must be read and transcribed to a messenger information sequence. This messenger information sequence is called the mRNA strand, and the process is called transcription. mRNA are linear molecules with an open reading frame that codes for protein sequence(s). A single stranded RNA is transcribed from double stranded DNA. Bacterial mRNAs can have one (monocitronic) or several (polycitronic) genes. This means that bacterial mRNAs can have information for more than one polypeptide on a sequence. The mRNAs are read as triplets (codons) from 5' to 3' of the DNA strand. The first codon is always AUG (Methionine), but for some bacteria GUG and UUG have been observed (Watson, 1965/2014).

The mRNA strand is translated into an amino acid by a process called translation. The ribosome is an enzyme that facilitates the translation from mRNA to protein by attaching itself on the mRNA strand. From there transporter RNAs or tRNAs (complementary to the codon in the mRNA transcript) elongates a growing peptide chain one codon at a time inside the ribosome. When a stop codon is reached, the elongation terminates leaving behind the finished peptide chain.

2.1.1 Promoters

Promoters are found upstream of a coding sequence. Their function is to facilitate the transcription from DNA to mRNA. In the prokaryotic genome only one factor protein (sigma) is involved in the initiation process of transcription. The sigma factors aid RNA polymerase to recognize promoters. There exist multiple sigma factors, where the sigma-70 is the most common one (Mejia-Almonte et al., 2020).

The promoters recognized by the sigma-70 containing holoenzyme are defined by two hexamer sequences. Namely the -35 and the -.10 boxes. They are separated by spacing region of +/- 17 nucleotides (nt). The consensus sequence for the sigma-70 has been determined to be consensus-sequence (1) (Brenner, 2001).

$$\text{TTGACA}–N_{17}–\text{TATAAT} \qquad (1)$$

From the middle of the -10 box to the middle of the -35 box the sequence (1) forms almost two complete DNA helical turns. The sequences are in other words located on the same side of the helix, and they are more easily recognized by the sigma factor of the holo RNA polymerase.

Table 2.1.1 Sigma factors of Escherichia coli retrieved from (Brenner, 2001). N_x indicates any nucleotide (N) x times.

Factor	Gene	Consensus binding site	Genes regulated
σ^{70}	*rpoD*	TTGACA–N_{17}–TATAAT	Housekeeping
σ^{54}	*rpoN (ntrA)*	CTGGCAC–N_5–TTGCA	Nitrogen metabolism
σ^{S}	*rpoS (katF)*	TTGACA–N_{12}–TGTGCTATACT	Stationary phase
σ^{32}	*rpoH (htpR)*	CTTGAA–N_{14}–CCCCATNT	Heat shock
σ^{F}	*fliA*	TAAA–N_{15}–GCCGATAA	Flagellar proteins
σ^{E}	*rpoE*	GAACTT–N_{16}–TCTGA	Extreme heat shock
σ^{fecI}	*fecI*	GGAAAT–N_{17}–TC	Iron transport

The promoters of the different sigma units have different consensus sequences. A consensus sequence is by definition the most frequent nucleotide or amino acid found at each position in a given alignment (Watson, 1965/2014). Examples of sigma factors present in *Escherichia coli* can be seen in table 2.1.1.1. These sequences are mainly conserved although some deviations exist. The sequences of binding sites are in other words not always the same.

2.1.2 The ribosomal binding site

The upstream of the start codon usually contains a purine-rich sequence that pairs with a complementary sequence in the 16S rRNA component of the small ribosomal unit (Kozak, 1999). Initiation of translation is regulated by the purine-rich sequence with the consensus 5'AGGAGG3'. The sequence is also called the Shine Dalgarno (SD) sequence. Various Shine Dalgarno sequences have been found in prokaryotic mRNAs. Common for them is that they lie around 10 nucleotides upstream from the AUG start codon (ThermoFisher, n.d).

The Ribosomal Binding Site (RBS) is located within the 5' untranslated region of mRNA and encloses the SD sequence, start codon and a short spacer in-between (Volkenborn et al., 2020). An example of an RBS can be seen in table 2.1.1 An Omatojo et. al argues that the length of the spacer enclosed in the RBS plays a role in the initiation of translation (Omotajo et al., 2015). Another article from 2020 estimated the optimal spacer length to be at least 7 to 12 nucleotides (Volkenborn et al., 2020). The results were sequence-dependent, and not a universal result.

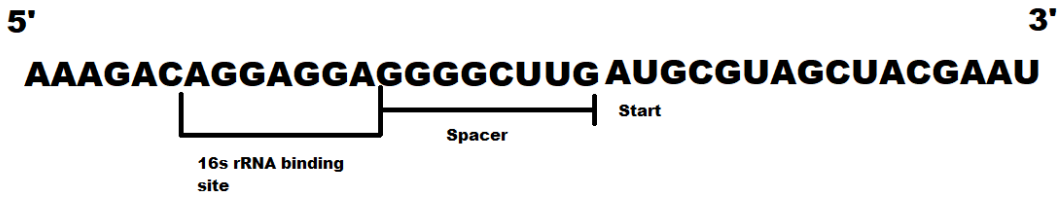


Figure 2.1.1 Illustration of the beginning of a typical mRNA transcript. The 16srRNA binding site is a complementary sequence of the 16S rRNA component of the small ribosome unit and is located upstream of the coding sequences' start codon.

As for the promoter sequences, the RBS is not always the same set of sequence. In fact, they are highly degenerated with a great variation in base composition and localization. Because of this, any conventional similarity search methods may have a very high error rate in their predictions (Oliveira, 2004).

2.1.3 The Operon

Some genes in the prokaryotic genome assemble in clusters called operons. These genes have the same promoter and terminator and are usually related either metabolically or functionally. The operons are usually under the control of a single promoter. This promoter is controlled by some regulatory elements called the operator that respond to external factors such as a substance concentration. Some operons have regulatory genes upstream of the operon that produce repressors. This regulator can either block transcription, leading to less protein product, or function as an activator when removed (Britannica, 2018).

The polymerase RNA-enzyme transcribes all the coding sequences present in the operon as a single RNA strand. Thus, only one ribosomal binding site upstream of the first coding sequence in the operon exists (see figure 2.1.2).

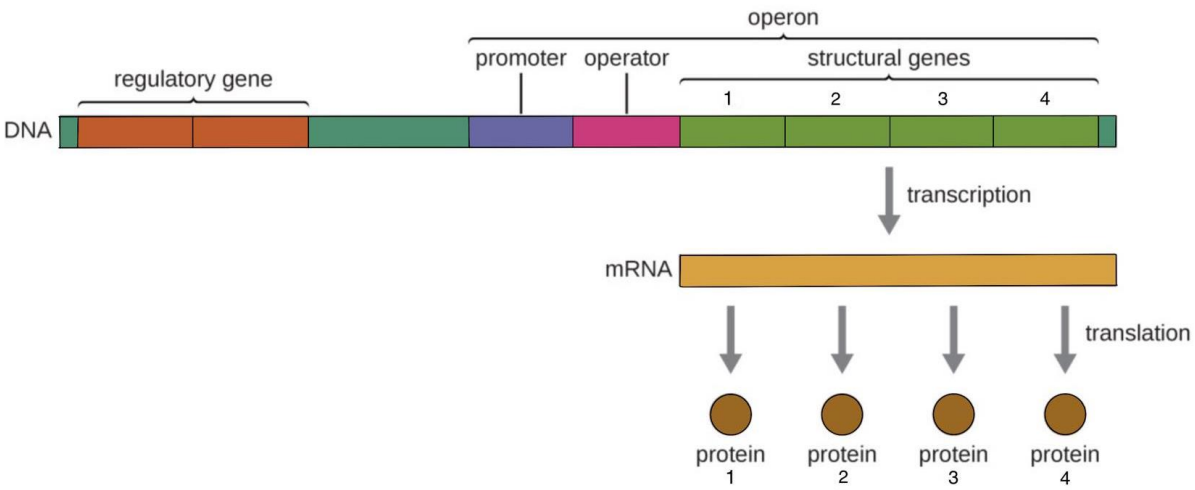


Figure 2.1.2 illustration of the operon clusters. The operon consists of regulatory genes that either activate or repress the transcription process. The transcript from the operon is a polycitronic template that when translated creates multiple proteins.

2.2 Open reading frames

Proteins are encoded in open reading frames (ORF). Contained within an ORF are a span of three nucleotides at a time between the start and stop codon (Mir et al., 2012). The open reading frame starts with a start codon (ATG, GTG or TTG) and ends with a stop codon (e.g., TAA, TGA or TAG). For every stop-codon in the genome there are usually many different start codons, and it gives rise to overlaps of ORFs.

DNA is partitioned into three nucleotides (codons) and contains two anti-parallel strands. Because of the nature of the DNA, there are six possible frame translations. For a coding gene however, only one is considered open.

In an open reading frame, there can exist multiple ORFs that share the same stop codon. These ORFs are also called nested ORFs. The ORF that is found the furthest upstream from the shared stop codon is defined as the “Longest Open Reading Frame” or LORF. This reading frame is the longest reading frame and has a start-codon found upstream of the other nested reading frames. An example of the overlapping sequences can be seen in figure 2.2.1.

5' ATGTCAAGTTTGAGAATGTCCCAGCACGAATGTGGCCAACTTAG 3'
3' TACAGTTCAAACTCTTACAGGGTCGTGCTTACACCGGTTGAATC 5'

Figure 2.2.1 illustrating a long open reading frame. Green letters are a potential START with a shared STOP (red).

Because of these nested ORFs, many software tools have issues with finding the correct start sequence of the genes. A study made in 2008 found that genomic overlap plays an important role in the annotation of genes, specifically for weak start codons (Palleja et al., 2008). Some argue that overlapping of ORFs is a way to reduce genome size (Cebrat et al., 1997). Overlap compresses the information into short sequences. However, it also makes exact prediction of prokaryotic genes difficult. Not all ORFs are coding sequences. These alternative ORFs can misguide some annotation software to give a false start point of a coding gene.

The correct annotation of the start codon is crucial. It is crucial because a correct mapping of the start codon in an ORF reveals accurate information about the proteome and can reveal important biological functions. The functions can give us a more complete picture of the organism, and in turn provide a better understanding of the organisms that exist around us. By better understanding the metabolism of a beneficial microorganism, the knowledge can aid us in improving its efficiency.

2.3 Sequence characterization - annotations

Currently there exists several different prediction software tools focused on prokaryotic genome finding. One of these tools is Prodigal, a genome annotation software that utilizes many elements like start codon usage, RBS motif usage and GC frame bias for gene prediction (Hyatt et al., 2010). Prodigal's focus when released was reducing false positives in prokaryotic genome annotation.

Genomes are usually spaced into coding and non-coding regions, where the issue lies in classification of those two. Currently the characterization of unknown sequences involves comparing it to known genomes or protein domains. This is usually done by comparing the sequences directly using external databases like BLAST or other homology search tools.

Identification of homologous sequences are mainly done using sequence similarity searching. The concept of homology is a common evolutionary ancestry and is central to computational analysis of proteins and DNA sequences (Pearson, 2013). Motifs are patterns that often repeats itself more often than expected. However, a pattern cannot be formed before multiple observations have been made, making homology search tools reliant on experimental data.

A separate method for annotation of genomes is finding genes from scratch. This method requires utilization of already established theories surrounding genes and transcription. Unlike searching for homologous sequences, one can search for ORFs, or map RNA-sequencing data to the genome. In many existing software, homology search methods and "manual" searching of genes are combined.

2.4 The RefSeq reference genomes

The current sequencing data consists of many contigs and few complete genomes as the sequencing technology is not yet fully perfected. The NCBI database has a wide range of genomes available, a subgroup of this being reference genomes. The reference genomes are assemblies annotated and updated by some submissions chosen by a curatorial staff in RefSeq (NCBI, 2021b). RefSeq uses tailored data models and consists of a single annotation pipeline called Prokaryotic Genome Annotation Pipeline (PGAP) (Haft et al., 2018). A reference genome is a collection of digital nucleic acid sequences stored in a sequence database. The annotation of these references requires the assembly of multiple databases that combines prediction algorithms and homology-based methods (NCBI, 2021b). The annotation of the genome is a lengthy process with many levels, and includes a prediction of protein-coding genes as well as other genome units like tRNAs, RNAs, pseudogenes, control regions, repeats, mutations, and mobile elements (NCBI, 2021a).

The most recent, well used algorithms for the classification of proteins to a given gene uses Hidden Markov Models (HMMs). The hidden Markov Model intelligently guesses the sequence of genes based on their different statistical properties. Each state (one for each label) has its unique emission probabilities stating the probability that it generates an A, C, G or T (Eddy, 2004). This, however, requires pre-based probabilities. NCBI is therefore constantly creating new HMMS and pipelines according to new findings in the field of genomics (NCBI, 2021b).

In 2016 Tatusova et al. in collaboration with Georgia Tech and NCBI created a new automated pipeline for annotation of genomes. This included alignment-based methods of predicting protein-coding and RNA genes, as well as other elements directly from sequence (Tatusova et al., 2016). Another update was made in 2018 by Haft et al. which included a new development of a hierarchical evidence scheme, as well as re-annotation of RefSeq prokaryotic genomes.

The annotation of the genomes in the prokaryotic reference genomes are not always complete, which is why it is continuously being worked on. The algorithm searches for potential genome sequences having start and stop codons, and the given sequence in between is then cross-referenced in a database for protein sequence similarity. An overview of the latest hierarchical workflow for annotation can be seen in figure 2.4.1.

For new proteins that the PGAP cannot name by any method, and proteins below 40% identity to protein-clusters, fall in an annotation called “hypothetical proteins” (Haft et al., 2018). This annotation is a zero-annotation with no value and small credibility. A better annotation would be hypothetical conserved protein which is an annotation that has been predicted but has a lack of experimental evidence for (Galperin, 2001). Whereas another step above would be an annotation like “putative” followed by a specific function.

The pipeline cross-references genome sequences to protein sequence databases. These sequence databases are updated regularly with new sequences. Pfam for instance, base its clustering on the MMseqs2 software (Mistry et al., 2021), making it heavily reliant on experimental data.

The decreasing cost of sequencing and increasing number of reads means it is important to automate the annotation progress. The current rate growth of sequencing data produced doubles approximately every 18 months (NCBI, 2022). This vast availability and increase of data leads to a big need for effective automation of genomic annotation.

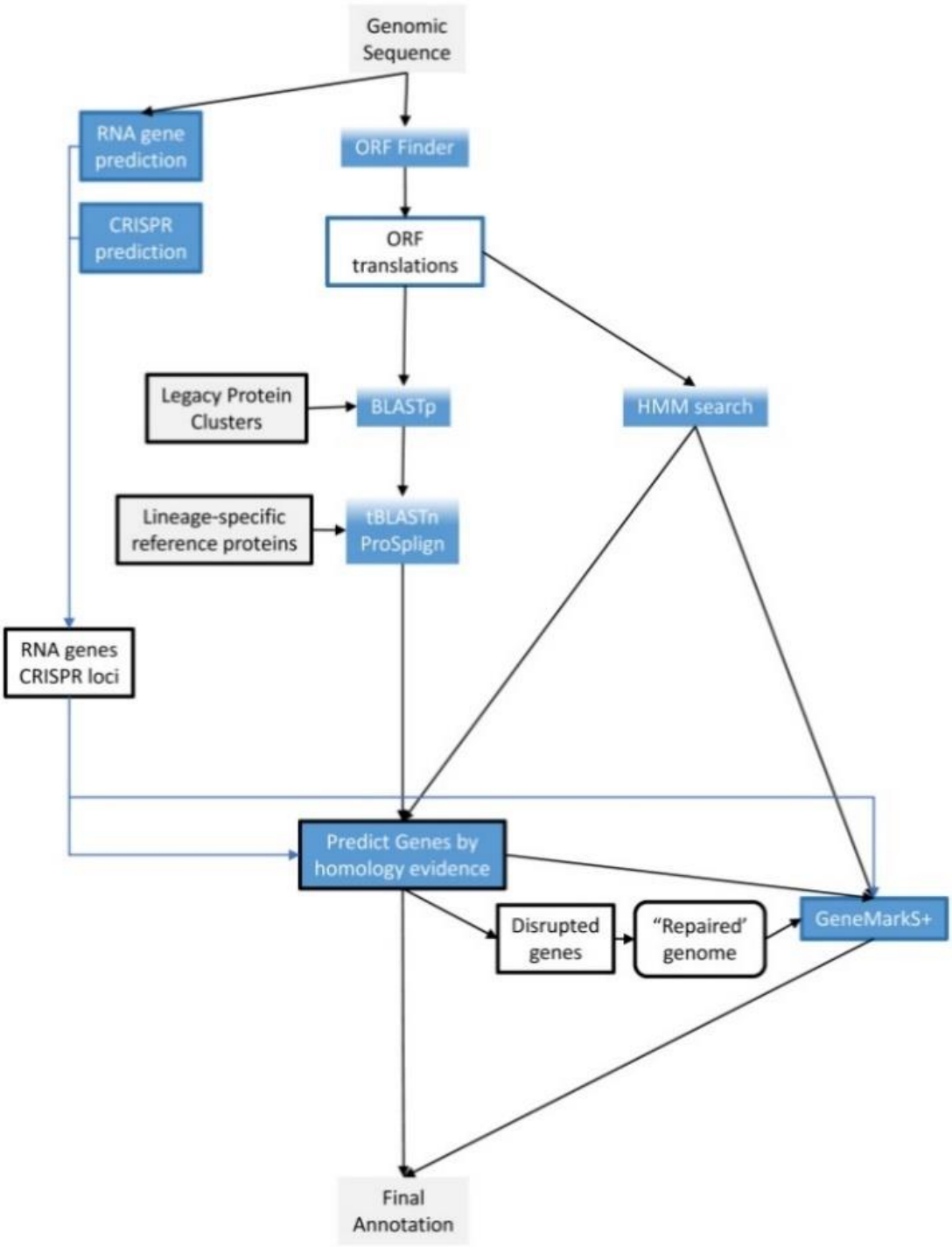


Figure 2.4.1 retrieved from (Haft et al., 2018). Newest workflow for structural annotation. Computational processes are shown in blue, data is in white or gray (Haft et al., 2018). It is important to note that this pipeline detects "disrupted genes" and some uncommon ones. According to (Li et al., 2021) GeneMarkS-2+ replaced GeneMarkS+ in PGAP November 2018.

2.5 Machine learning

With the increase of sequencing data, there is an increased need for automation of analysis. Machine learning is a process that allows for automation of analytical model building. A machine learning method can learn from data and apply what they have learned to new data.

2.5.1 Classification

Classification is the process of categorizing something into a certain group based on some characteristics. An example of classification is discerning whether a person is ill or not. In this example we have characteristics such as temperature and blood pressure. This is dependent on the sickness we are dealing with. For annotation of genes, we usually discern between gene (0) and no gene (1). This type of classification is called a binary classification. Some classifiers have continuous output, for instance a probabilistic output estimating a target probability. Other models produce a discrete output that only indicates the predicted class.

2.5.2 Preprocessing

There are many things to consider before feeding a dataset into a classifier. The data fed into a machine learning model must consist of numerical values for the model to retrieve any information. Text and sequences cannot be fed directly into a machine learning model and needs to be preprocessed. There are many ways to process text, some examples are dummy encoding and K-mer count.

In dummy encoding a set of categorical variables are converted into binary variables (also called dummy variables). A nucleotide sequence of length 2 for instance, would result in a matrix consisting of 4 * 2 variables. The output given would be a binary output (0 or 1) indicating at which position the nucleotide was found. If we have a list of 3 sequence observations (AG, TA, and GC) the dummy encoded matrix would appear as seen in table 2.5.1.

Table 2.5.1 an example of dummy-encoded sequence data for the three observations AG, TA and GC. The column names indicate the nucleotide base as well as their possible locations in the sequence.

Sequence	A_1	A_2	C_1	C_2	G_1	G_2	T_1	T_2
AG	1	0	0	0	0	1	0	0
TA	0	1	0	0	0	0	1	0
GC	0	0	0	1	1	0	0	0

K-mer retrieval and analysis from sequences are quite common in nucleotide sequence analysis. The frequency of K-mers are often of great importance when attempting to differentiate between certain genes according to their codon-bias (Iriarte et al., 2021). K-mer frequencies can also be applied on other genomic fragments, as nucleotide composition varies a great deal between genomes (Perry & Beiko, 2010).

K-mers are substrings of a string with a length *k*. A sequence of length *L* will have $L- k + 1$ K-mers and a total of n^k possible K-mers where n is the number of possible monomers (four monomers in DNA). The total number of possible K-mers increases with an increase of sub-string length (*k*). An example of different K-mers for the sequence ACTGAATCC can be seen in table 2.5.2.

Table 2.5.2 an example of K-mer sequence data for the sequence ACTGAATCC. Shown here are monomer to five-mer

K	K-mer
1	A, C, T, G, A, A, T, C, C
2	AC, CT, TG, GA, AA, AT, TC, CC
3	ACT, CTG, TGA, GAA, AAT, ATC, TCC
4	ACTG, CTGA, TGAA, GAAT, AATC, ATCC
5	ACTGA, CTGAA, TGAAT, GAATC, AATCC

When dealing with DNA sequences a K-mer of 6 gives rise to 4^6 different combinations or 1024 combinations. For a potential training data this means 1024 features if only 6-mers are considered. The shorter the sequence at hand, and the longer the K-mers, the more zero-sparseness will be observed. For instance, A 6-mer nucleotide count on a sequence of 30

gives rise to 25 possible K-mers per sequence. With the number of unique 6-mers at 1024, at least 999 possibilities have a count of zero.

Highly correlated features may appear and are often redundant. For these cases a dimensionality reduction technique may be applied, and ultimately leads to the model running faster. The reduction can also decrease the signal-to-noise ratio (Raschka, 2019).
For multivariate data, sometimes a features variance is in a much larger scale than another feature in the same dataset. The features may for instance have been measured in different units of measure. The element that has a larger scale may dominate other elements in the dataset and may lead to biased prediction outcome for many machine learning methods. A solution to the different scaling of a multivariate dataset would be to scale the data prior to modelling. A scaling technique commonly used is standardization (2). This scaling would be done column-wise.

$$Z = \frac{X - \mu}{\sigma} \qquad (2)$$

In (2) the *X* is the specific value or observation. *Mu* (μ) is the mean of the variable and sigma (σ) is the standard deviation. Calculation of a standardized value is the same as finding the z-score.

Standardization allows for comparison between different types of variables. The standardization technique creates variables with a mean of zero and a unit variance of 1 (Raschka, 2019).

Raw data is seldom optimal for training a learning algorithm. Preprocessing data is a crucial step in any machine learning application before moving on to the modelling part.

2.5.3 Hyperparameter optimization and cross validation

A machine learning algorithm's objective is to find a function that best explains some samples that follow a grand truth. Very often, a learning algorithm produces the function through an optimization of a training criterion with respect to a set of parameters (Bergstra, 2010).

A hyperparameter is a parameter one can change before model training. The tunable parameters are manually set, meaning they are iterated through and compared using a certain

metric (see Section 2.5.10 evaluation for more about metrics). The most common way of performing a hyperparameter tuning is through a grid search. A grid search is a systematic search of all possible hyperparameters of a parameter by training and testing the machine learning algorithm. The training and testing can be measured by cross validation on the training set or evaluating a hold-out validation set. (Chicco, 2017).

Cross validation is a resampling method that randomly partitions data to test and train a model. The data is partitioned into *k* folds where the training of a model is performed on the *k-1* partitions. The remaining partition is tested after training. The training and testing are performed *k* times where the resulting metric is presented as the mean of all *k* runs (Arlot, 2010). See figure 2.5.1 for an illustration of a cross validation.

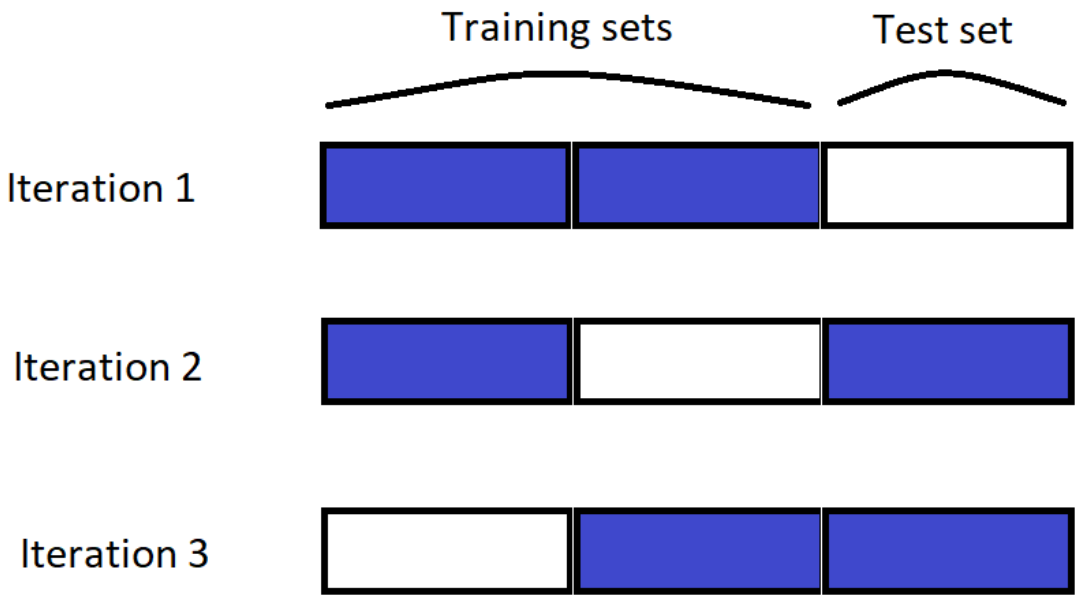


Figure 2.5.1 an illustration of a 3-fold cross validation. The training data is here divided into three sets.

2.5.4 Logistic Regression

Logistic Regression is a commonly used model in statistics. This regression model estimates the probability of one event (out of two) by using the logarithm of the odds. It is one of the most widely used algorithms for classification in the machine learning industry (Raschka, 2019). In general, it performs well on linearly separable classes. For a binary classification, this is a linear model.

The logistic regression model is a probabilistic model for binary classification. This means the output given is a continuous variable giving a probabilistic output of how likely it is that an observations falls under class 1. For the activation function (the logistic regression function) to fall into a binary classifier, the probability is converted using a threshold function. A common threshold is one where results below 0.5 fall into class 0 and results above 0.5 falls into class 1.

2.5.5 Partial Least Squares reduction and Linear Discriminant Analysis

Partial Least Squares Analysis (PLS) is a multivariate dimensionality reduction tool (Wold, 2001), and an adaption of PLS regression methods for supervised clustering. The PLS aims to maximize covariance between independent variables and class information. It is particularly useful for multivariate datasets.

The features created are referred to as principal components (PC) in PCA and just components in PLS (Pearson, 1901). In PCA the first PCs contain as much variance as possible, whereas PLS preserves as much covariance as possible between the original data and its labelling (Ruis-Perez, 2020).

The LDA method aims to find a linear combination that can separate two or more classes. The combination found could then be used as a linear classifier. LDA's primary purpose is to project high-dimensional data onto a low-dimensional space whilst achieving maximum class separability (Barker, 2002).

PLS is a common feature reduction tool and is often used in preprocessing data. For a two-class problem the PLS dimensionality reduction is to be preferred over PCA (Barker, 2002; Liu, 2007). After dimension reduction using PLS one can then use LDA in the truncated score-space for classification.

2.5.6 K-nearest neighbors

K-nearest neighbors is dubbed a lazy learning algorithm. This is because it does not learn a discriminative function from the training data. Instead, it memorizes the training data set. Based on the training data, any new point introduced to the model is assigned to a class based on majority voting among its nearest neighbors (Cover, 1968). The number of nearest neighbors used is tunable. For the nearest neighbor parameter this means the number (*n*) neighbors that an observation is compared to is a manual input. Same applies to the distance calculation, where one can either choose a Minkowski, Manhattan, or a Euclidian distance calculation. See figure 2.5.2 for an illustration of the difference between these distances.

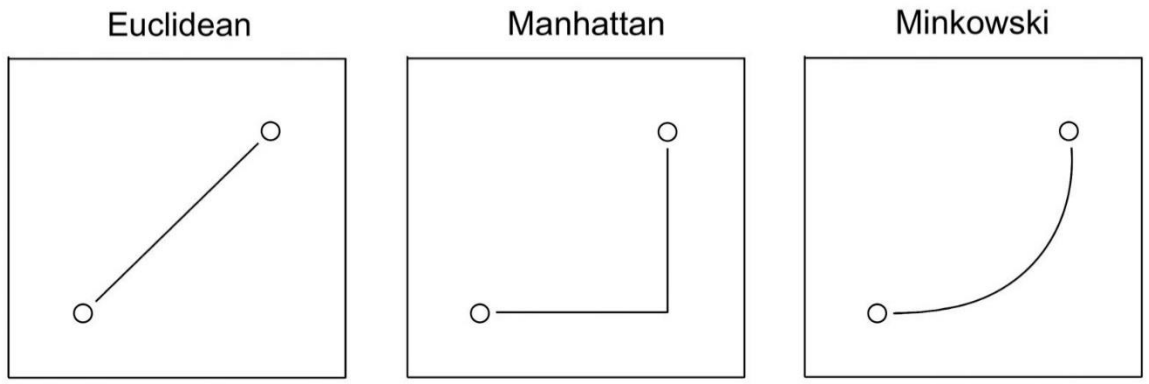


Figure 2.5.2 illustration of the path taken for computation of the distance between two points for the three methods; Euclidean, Manhattan and Minkowski.

The Minkowski can be thought of as a generalization of the Euclidean and Manhattan distance. The distance calculations (Euclidean and Manhattan) are part of an exponent in the Minkowski-formula. The Minkowski distance is typically used with the exponent *p* set at either 1 or 2, which correspond to the Manhattand and Euclidean, respectively (Gabbay, 2005). The illustration of Minkowski seen in Figure 2.5.2 is a *p* of around 1.5.

2.5.7 Decision Tree

Decision tree is a supervised learning method used for both classification and regression problems. The decision trees are based on the best way to split the observations into their target groups based on the features given. When it is time to split a node in a decision tree, every feature is considered before choosing the feature that causes the best separation between the different classes, this is called the information gain or entropy (Raschka, 2019).

Decision trees start at the root node where all the information from the dataset is stored and is from there split into branches. An illustration of a decision tree can be seen in figure 2.5.3 Each decision node represents a test on an attribute where each branch represents the outcome of said test. Each leaf node represents a class label. In summary, the path from root node to leaf node represent a set of classification rules needed to classify the observations of a dataset.

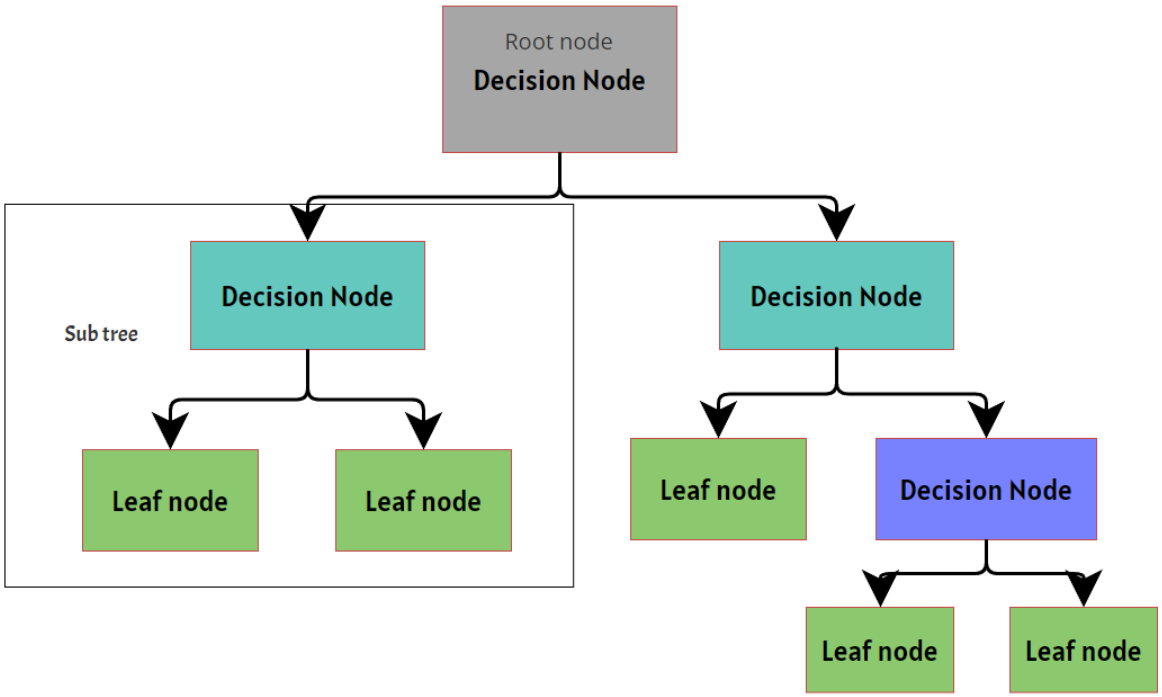


Figure 2.5.3 Illustration of the topography of decision tree. Each decision tree consists of a decision node that branch either into leaf nodes or decision nodes.

Typical hyperparameter for a decision tree classifier is max depth. The max depth is here the max depth a tree can have. If max depth is infinity, the nodes are expanded until all leaves are pure, that means the samples split contain only a given target (sklearn, N.Y).

2.5.8 Random Forest

The random forest ensemble is built on creating many decision trees (with number of trees being a hyperparameter). For random forest, only a subset of these features, as well as a subset of samples, are chosen at a time. This forces more variation to pass during training and leads to lower correlation across trees and more diversification (Breiman, 2001). By averaging all the trees in a random forest, the final model can discern better in the final

feature subspace. The information about which features are most important can thus be retrieved after ensemble.

For an imbalanced dataset the training in Random Forest may be biased towards the majority classes. A way to counterbalance this is by randomly down sample the majority class to the same number of samples as the minority class. This forces random forest to negate any unnecessary variation that occurs in the majority class and allows it to discern between the majority and minority class potentially better. This method of Random Forest is also called a Balanced Random Forest.

In the random forest ensemble, the trees have split features based on information gain. If there is high correlation however, one feature may be ranked highly, and the relationship may not be fully captured (developers, 2007-2022).

2.5.9 Gaussian Naïve Bayes

The gaussian naïve bayes is a classification model that assumes normally distributed data. The gaussian naïve bayes takes the training data sets' and estimates the mean and standard deviations for each variable according to their target label (for binary classification, label 0 or 1) (Webb, 2005).

When classifying a new observation, the model considers the log (base e) of the prior probability of classification as well as the log-likelihood of variables present. The calculation yields the posterior probability of belonging to a given class (Minsky, 1961). The lower the log score, the less likely the observation belongs to the given class.

2.5.10 Evaluation

When training a model, the machine learning methods used usually have many different parameters. An example can be different distance measurements. These parameters are called tunable parameters and is adaptable to each training data. To find the best parameter, we need to have a measure of closeness. One can create a contingency table called a two-class confusion matrix (table 2.5.1). This matrix shows the elements that are correctly predicted, and which were not. This allows the models trained to have a universal comparison of fit. This is comparable not only when training the same model with different parameters, but also for the comparison of different models.

Table 2.5.1: General confusion matrix. The classification is correct if both the predicted and actual classes are positive or negative.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Precision is defined as a measure of quality (3).

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

Where TP is the total true positives and FP is the total false positives.

Recall is defined as a measure of quantity where the formula is (4)

$$Recall = \frac{TP}{TP+FN} \quad (4)$$

Here FN is the number of false negatives and TP is true positives.

A balance between the precision and recall metric is the F1-score (5). It is deemed as the harmonic mean of precision and recall. Its values range from 0 to 1, where 1 is perfect classification.

$$F1\ Score = \frac{2\ x\ (Precision\ x\ Recall)}{Precision+Recall} \quad (5)$$

When evaluating binary classification with real numbers ($\in$ R) a cutoff threshold is needed to discriminate between positive and negative classification. Commonly, the confusion matrix is computed for all possible cut-offs and then these matrices can be used to create a Receiver Operating Characteristics (ROC) curve.

The Receiver Operating Characteristic or ROC curve is a plot of true positives versus false positive rates at some classification thresholds. The true positive rate is here the recall, and the false positive rate is 1 – specificity (6).

$$Specificity = \frac{TN}{TN+FP} \quad (6)$$

The ROC curve shows the connection between recall and specificity for every possible cut-off for a classification model. This allows a graphical overview for finding a classification threshold suitable for the issue at hand (Fawcett, 2006). An illustration of an ROC graph with 4 discrete classifiers can be seen in figure 2.5.4.

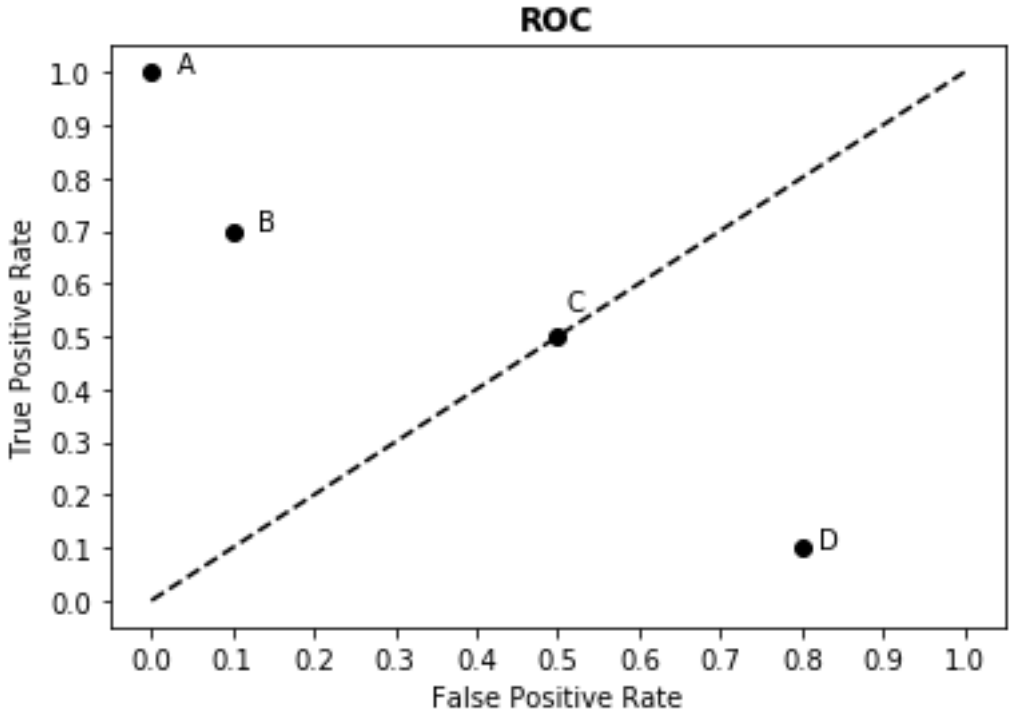


Figure 2.5.4 a basic ROC graph showing 4 discrete (binary) classifiers

Classifiers on the left-hand side of an ROC graph are thought of as conservative. This is because they make positive classifications with strong evidence resulting in few false positive errors. However, it results in a low true-positive rate as well. Classifiers on the upper right-hand side often make positive classifications with weak evidence, meaning they classify nearly all positives correctly, but with high false positive rates. The latter are deemed as liberal.

In figure 2.5.4 the diagonal line shows the strategy of randomly guessing a class. At the point C (0.5, 0.5), C's performance is virtually random, guessing the positive class 50% of the time. Point C can be thought to have no information about the class, whereas any classifier that falls

below the diagonal may be said to have useful information, but is not applying it correctly (Flach, 2002). By reversing point D's classifications, it produces a point in the upper left triangle.

The area under the ROC curve or AUC measures the two-dimensional area underneath the ROC-curve. It ranges between the values 0 to 1. The area is useful for comparison of different classifiers, but also yields the probability that a random observation of the positive class is ranked higher than a randomly chosen negative instance (Hanley, 1982).
The F1, recall and precision metrics only include three out of four confusion matrix categories (TP, TN, FP). With a highly imbalanced dataset where the positive class is in minority, a small change in the positive direction may shift the score significantly. Matthew's correlation coefficient (7) measures the correlation of the true positive classes c with the predicted labels l:

$$MCC = \frac{Cov(c,l)}{\sigma_c \sigma_l} = \frac{TP*TN - FP*FN}{\sqrt{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)}} \quad (7)$$

Where the worst value is -1 and the best value is +1. Cov(c,l) is the covariance of the true classes c and predicted labels l whereas σ_c and σ_l are the standard deviations respectively (Chicco et al., 2021). TN is here the number of True Negatives. In a software like prodigal, the focus lies solely on the positive rate, and thus it is not possible to retrieve the true negative rate, making it difficult to classify its true correlation.

3. Method

This chapter will present the steps taken from retrieval of raw data to the running of a machine learning model. The methods described here are based on existing theories described in Section 2.

Data analysis and wrangling was carried out using Rstudio 4.1.0 (R Development Core Team, 2010). Machine learning was carried out using Scikit-learn in Python 4 (Pedregosa, 2011), except for PLS+LDA that was carried out using the *mpda* package in Rstudio 4.1.0 (Snipen, 2017). Some figures are made with the *ggplot2* package in R (Wickham, 2016), and some with the *matplotlib* package in Python (Hunter, 2007).

3.1 Data

The retrieval of raw data is an imperative first step towards making a training dataset. This section will present the gathering of annotated genomic data for use in the final training dataset, as well as some qualitative processing.

Data was downloaded from (NCBI, 2021c). RefSeq FTP files with data containing the genomic sequence was downloaded for each reference sequence in the database (15 in total), an overview of the different genomes can be seen in table 3.1.1. The genomes presented will be referred to by their respective species name. For supplicate species, the strain is attached to separate them.

Table 3.1.1: Overview of the reference genomes from the RefSeq prokaryotic database. The respective columns are organism name, their genomic size in megabytes, their GC base content in percent, how many scaffolds there are in the genome, and the number of coding sequences (CDS).

Organism Name	**Size (Mb)**	**GC%**	**Scaffolds**	**CDS**
Campylobacter jejuni subsp. jejuni NCTC 11168 = ATCC 700819	1.64	30.5	1	1572
Salmonella enterica subsp. enterica serovar Typhimurium str. LT2	4.95	52.2	2	4548
Staphylococcus aureus subsp. aureus NCTC 8325	2.82	32.9	1	2767
Listeria monocytogenes EGD-e	2.94	38	1	2867
Mycobacterium tuberculosis H37Rv	4.41	65.6	1	3906
Escherichia coli str. K-12 substr. MG1655	4.64	50.8	1	4285
Shigella flexneri 2a str. 301	4.83	50.67	2	4313
Pseudomonas aeruginosa PAO1	6.26	66.6	1	5572
Chlamydia trachomatis D/UW-3/CX	1.04	41.3	1	888
Coxiella burnetii RSA 493	2.03	42.64	2	1833
Bacillus subtilis subsp. subtilis str. 168	4.22	43.5	1	4237
Klebsiella pneumoniae subsp. pneumoniae HS11286	5.68	57.14	7	5779
Caulobacter vibrioides NA1000	4.04	67.2	1	3886
Acinetobacter pittii PHEA-2	3.86	38.8	1	3599
Escherichia coli O157:H7 str. Sakai	5.59	50.4	3	5155

General feature formats (GFF files) were also downloaded for the 15 reference genomes. GFF describes the genes and contains nine informative columns for coding sequences. The GFF data was decompressed for all genomes. Using GFFread from Microseq version 1.0, the annotated data was read and converted into a table with 9 columns. Information about the columns can be seen in table 3.1.2.

Table 3.1.2 Information from the nine columns in any given GFF file. Description retrieved from (Stein, 2006).

Column name	**Description**
Seqid	Name of the chromosome or scaffold
Source	The software that generated the feature
Type	Type name, contains a Sequence Ontology (SO) accession number.
Start	Start position of the feature
End	End position of the feature
Score	Floating point value.
Strand	Either + (forward) or - (reverse)
Phase	Indicates where the next codon begins relative to the given CDS feature. Contains integer 0, 1, 2 where 0 indicates a codon beginning on the first nucleotide.
Attributes	A list of feature attributes, among them a string of protein information

The columns Start, End, Seqid, Type and Attributes were selected for future use. Attributes were then filtered so that only the protein product remained listed as a string. The final table was dubbed the reference GFF file and would be the reference for all coding sequences present in all genomes. For it to be comparable genome-wise, another column was added, namely the genome column. This column reveals which rows belongs to which genomes.

All the GFF files downloaded from RefSeq consists of 9 variables and a summed total of 55 538 observations, each observation being details about a protein product from a given reference genome.

After importing and sorting of the table, counting of uncertain proteins was implemented. The counting was performed to assess the quality of the raw data. Uncertain proteins are here defined as all the observations with the words “putative”, “hypothetical”, and “hypothetical conserved” attached to their attribute column.

3.2 Comparison with a gene prediction software

It was desirable to compare the RefSeq annotations with Prodigal predictions of the reference genomes. This comparison was made to illustrate a standard software's issue in locating the correct start-codon. After using the genome fasta files for prodigal prediction, a common signature for each reference-genome was created. The signature consisted of sequence-ID, end-position for the gene, and strand type. These signatures were then compared to the reference-annotations using the precision and recall metrics. The comparison was made genome-wise.

After the first comparison, an inquiry was made to compare the prodigal-annotation and reference-annotation with a signature having both the start and stop location. The same metrics as for the signature containing Seqid, strand and end was used (precision and recall). A figure showcasing the two different signatures and what they illustrate is showcased in figure 3.2.1.

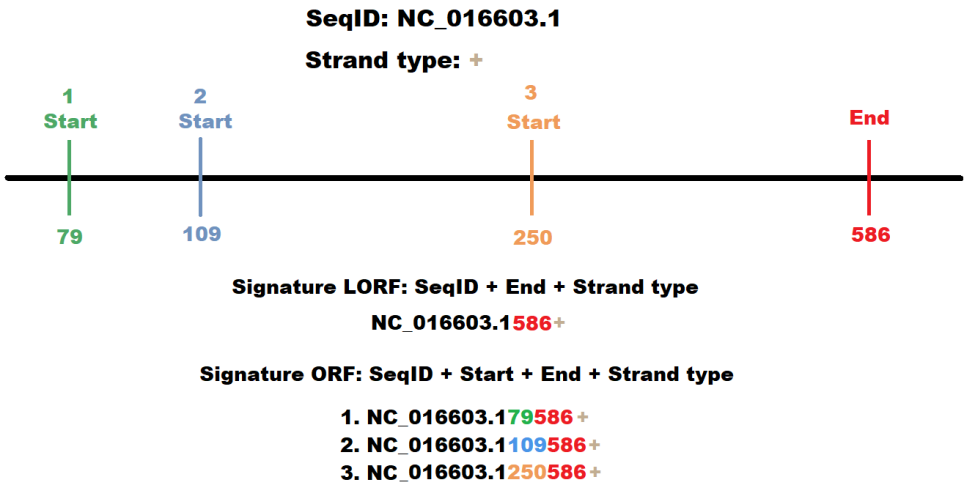


Figure 3.2.1 illustrates signatures used for similarity measurements for a positive strand. The LORF signature consists of 3 different concatenated attributes and the ORF signature consists of 4 different attributes. The red font is an indication for end position, black is sequence ID and brown is strand type. The different start positions contain differrent colors to separate between them.

To compare the ORF mapping with the prodigal output, the prodigal GFF files were further filtered to only include the LORF predictions that match the reference LORF. The attributes column in the GFF file contains information about whether an RBS motif has been found upstream of the potential CDS. It was desirable to compare the number of ORFs that had an RBS motif, both coding and non-coding, in the same LORF as a reference CDS.

3.3 Modelling

Using information from an upstream sequence, we wanted to classify between gene (target 1) and no gene (target 0) with respect to the different start positions present in a long open reading frame (LORF). To classify a sequence of letters, it needs to be converted to numerical data. For this data, the upstream sequences retrieved after ORF-mapping and comparison was converted into K-mer count data and one-hot encoded data. One-hot encoded data is from here on dubbed “sequential data”.

3.3.1 ORF-mapping and comparison

Another method of annotation is to search for open reading frames and investigate whether a pattern or motif can be enough to distinguish between a coding sequence and a non-coding sequence. The general idea between retrieval of ORFs and not LORFs is to be able to discern between the alternative start codons in a long open reading frame. By fetching all the alternative ORFs present, the hope for a machine learning model is to find a pattern between the coding and non-coding ORFs. By finding a pattern, a more precise annotation with respect to the start position of genes can be achieved.

The microseq package in R has a built-in function called *findORFs* that scans through a FASTA file. The function locates "Open Reading Frames" or ORFs in a fasta file. When locating the ORFs the function will find all subsequences in the fasta sequences that start with a start codon (ATG, GTG or TTG) followed by a number of triplets (codons) and ending with a stop-codon (TAA, TGA or TAG) (Snipen & Liland, 2017). The *findORFs* function was used to scan through the 15 reference .fna files. As in 3.2, a common signature was created for both the ORF and the reference annotation to investigate the precision and recall for the ORF-hits. The signature is a concatenation of sequence ID, strand type, and end position. Assumption a priori is that all genes are an ORF. A different number (50, 90, 150) of minimum ORF lengths were tried out.

If all ORFs had been included in the final dataset we would have a more skewed target balance between gene coding ORFs and non-gene coding ORFs, where all the non-coding ORFs would have been in majority. Therefore, the focus was shifted to alternative ORFs for the reference annotations. This meant filtering all alternative ORFs in the reference longest open reading frame. Only the ORFs that share the same LORF as a CDS in the reference file is included in the final dataset.

The ORF-hits table was further annotated with a target variable declaring if there was a match with the reference annotation genome-wise or not. For all ORFs in the respective genome a sequence of 30 bases upstream was found. It was desirable to compare the sequences of each ORF to see if there was a way to classify the start position of genes based on their upstream sequence.

The upstream sequences also had a variant with the start codon included as it may give a bias toward start-codons in coding sequences. This in combination with the different ORF length was tried in model training.

3.3.2 Upstream sequence retrieval

There are multiple elements surrounding upstream data that could be beneficial in a machine learning model. Ribosomal binding sites and promoters are said to be present upstream of some coding genes as presented in Section 2.1.1 and 2.1.2. The idea behind upstream sequence retrieval for every ORF is a test to see if the information present is enough to separate the coding and non-coding ORFs.

After obtaining and assigning target variables to the nested ORFs present in a LORF, the columns start, end and strand was used to fetch upstream data. A custom function called "Upstream finder" was used to achieve this. The function reads the specific genomes' fasta file and retrieves a self-defined number of bases upstream of all observations in a data frame with a GFF format.

The function has four arguments. The first is the given GFF data frame from which you have the given start and end position of a possible open reading frame. The second is the genome sequence itself, input here is expected to be a path to a fasta file. The third argument, length, is the number of bases upstream of the start position desired. Input is here an integer. The last argument called orf_bases inputs an integer, this argument indicates how many bases in the ORF sequence that is included. If integer is set to three, the start codon is included in the final upstream column. The default for the orf_bases argument is zero. The output from the function is a column containing the upstream sequence.

The functions first step is to read a given fasta file. This was achieved using the function readFasta() from microseq (Snipen & Liland, 2016). The function returns a table containing two columns of text. The "header" that contains header lines, and then the sequence itself.

Afterwards the GFF data frame defined is filtered according to strand type. The strand type is a key factor as the "End" position in strand type "- "indicates the start position of the open reading frame, and vice versa. From there the start position was set at 30 bases upstream from

the open reading frames start position, and the end position was set at one base upstream of the start position.

With the new start and end defined, the function Gff2fasta from microseq (Snipen & Liland, 2016) was utilized. This function takes the start and end position of a general feature formatted table (GFF) and retrieves the sequences in-between from a fasta file. The sequence retrieved here will then be in a fasta format, existing of the column's header and sequence. In the upstream_finder function the column "sequence" is returned.

3.3.3 K-mer data

K-mer count data of a sequence can be useful for discerning between target groups. The idea is that the mean count and variance is different enough between targets. The difference may allow any model to retrieve useful information during training.

The sequence data from 3.3.2 that was made for all open reading frames was used to create K-mer data. The creation of K-mer data was achieved using the KmerCount function in the microclass package (Vinje et al., 2016).

Different K-mers were tried out. The sequence length and four possible nucleotides gives 4^K possible combinations of K-mers affecting runtime and memory usage. An upper cap was therefore set at K = 6, to capture all the variance in the dataset, and without affecting computational time excessively. As there are only four unique letters present in the sequence data, a low K-mer count may not be as helpful information-wise. The lower cap was consequently set at K = 3. The different K-mers were assembled into a common feature space. Further on, because of the natural genomic variation present in the prokaryotic genomes, one K-mer dataset was created for each unique genome.

The frequency of K-mers was overdispersed with a variance larger than the mean, and the feature space was vast. The initial data for the *Escherichia coli O157:H7 str sakai* K-mer data had a total of 79 504 rows and 5442 columns.

The final training dataset then consisted of 3 to 6-mers of the open reading frames' upstream sequences located 30 bases upstream as well as the variable length and the target class. The

variable length was added to the final dataset as it may be a significant explanatory variable in classification.

3.3.4 Sequential data

After the initial training and creation of the K-mer data, another possible training data was made. The initial K-mer dataset contained information in form of counts of K-mers available upstream of ORFs but does not contain information about the order of the nucleotides. To capture the sequence structure, the sequence data was one-hot encoded creating a total of 120 features. This training dataset thus consisted of 120 features (4 bases possible in 30 different positions: 120 different combinations) as well as the target class variable. This dataset is much smaller than the K-mer dataset and is therefore computationally cheaper.

3.3.5 Classification Models

There exists a multitude of classification models and thus many ways to classify data. Based on this, seven models were run with tuning to find the better model. The models in question are seen in table 3.2.1. To account for result randomness during hyperparameter tuning, each model was run with a cross validation of 3.

To get an overview of different classification models performances, they were run on one genome. *Escherichia coli O157:H7 str sakai* had the highest number of coding sequences as seen in Table 2.1.1, as well as the genome with the median GC content, and was therefore a first choice for model training.

Table 3.3.4.1: Overview of the different classifier models used to distinguish the upstream sequence data between gene and no gene. The classifiers description as well as tunable parameters are defined her.

Classifier	Description	Tunable parameters
Logistic regression	Classification algorithm that predicts the probability of a categorical dependent variable. Predicts P(Y=1) as a function of X.	Regularization strength, penalty (l1, l2)
Gaussian naïve bayes	For continuous data, useful for high dimensional data.	Variance Smoothing
K neighbors classifier	Distance based lazy algorithm. Places all observations in n-dimensions and finds the k – nearest neighbors using a likeness score. The class is given by majority voting.	Number of neighbors and distance metric (Manhattan, Minkowski or Eucledian)
Decision tree classifier	Uses information from features and target in a dataset to discern where to best split the data in different classes.	Max depth
Random forest classifier	Ensemble classification method consisting of many decision trees.	Number of trees, max depth, and max features
Balanced random forest classifier	For imbalanced data, this classifier undersamples the majority class randomly to match the number of samples in the minority class. This is done for each decision tree.	Number of trees, max depth, and max features
PLS+LDA	LDA is a common feature reduction technique often paired with PLS for categorical data	Number of components for PLS

All the model trainings performed, using the classifiers seen in table 3.2.1, were trained using the scikit-learn tool (Campbell). With the exception of PLS + LDA that was run using the mpda R package (Snipen, 2017). The validation of the models was performed using an independent dataset (test-set). A prediction of the test dataset was run with all trained

classifiers seen in table 3.2.1. From the predictions the resulting false positive and true positive rate were plotted together in an ROC curve for visualization of all classifiers.

The creation of the test-set was achieved by randomly partitioning 25% of the data's observations into the test set. This left 75% of the data for training.

After the initial overview of classifiers, four common classifications were run on four selected genomes separately. These genomes were deemed to best represent the variety in genome content, with respect to their GC%. See table 2.1.1 for their respective GC%. The low GC% representative is *Campylobacter jejuni*, medium representative is *Escherichia coli O157:H7 str. Sakai*, and high GC% representative is Caulobacter *vibriodes*. The last genome; *Bacillus subtilis subsp. subtilis str. 168*, was selected due to their status as a model organism of the gram-positive lineage (Errington & Aart, 2020). The classifications were tuned for each genome, as we already assume they are independent.

The classifications in question are the Balanced random forest classifier, logistic regression, PLS + LDA and KNN. The balanced random forest was chosen due to the datasets imbalance and because the initial classifier showed promising results. The remaining three classifiers were chosen as they are very simple, but popular classifiers.

Finally, the classifier that had the highest ROC score was further utilized by training a model for all RefSeq genomes. The training on all genomes was done to get an overview of the variation between the 15 genomes present, as well as to see if the initial result was representative.

3.3.6 Feature Selection

The number of features with all K-mers available, plus the variable length, was at 5457. To reduce the complexity of a model and avoid overfitting, feature selection was applied. The feature selection does not only seek to avoid overfitting but can also give valuable information about feature importance.

To check for feature importance in the K-mer dataset two methods were run, namely random forest feature permutation and chi squared test.

3.3.5.1 Random Forest

The scikit-learn package has a permutation-importance function with a trained random forest model as input. The function shuffles features randomly during prediction and computes changes in its performance. The features that impact performance the most are deemed to be the most important ones. After tuning a Random Forest classifier, this function was run on the *E. coli O157:H7 str. Sakai* K-mer dataset.

3.3.5.2 Near-zero variance predictors

Variables with very few numerical values can cause errors or unexpected results. Near-zero variable predictors are predictors with very few unique values. This is very common in a K-mer dataset as well as a sequential one.

Table 3.3.6.1 Overview of number (n) of observations per genome in the final dataset with an ORF-length of 90 or above.

Genome	**n**
Acinetobacter pittii	46078
Bacillus Subtilis	55581
Campylobacter jejuni	17626
Caulobacter vibriodes	55681
Chlamydia trachomatis	12491
Coxiella burnetiid	24882
Escherichia coli K-12	68620
Escherichia coli O156H7 sakai	79504
Klebsiella pneumoniae	81887
Listeria monocytogenes	34993
Mycobacterium tuberculosis	80707
Pseudomonas aeruginosa	93489
Salmonella enterica	70091
Shigella flexneri	60889
Staphylococcus aureus	28597

The number of columns present, as well as the overrepresentation of 0’s, may make it difficult for any classification model to retrieve useful information from training. Therefore, any

columns with varying sums less than a given quantity (500, 1000 and 1500) were attempted removed to see if it affects the final model training. An overview of the number of observations per genome can be seen in table 3.3.6.1. Numbers were chosen according to the number of observations present in the minimum genome (*Chlamydia trachomatis*). The idea behind the removal is that the variables that fall below that number are useless in modelling.

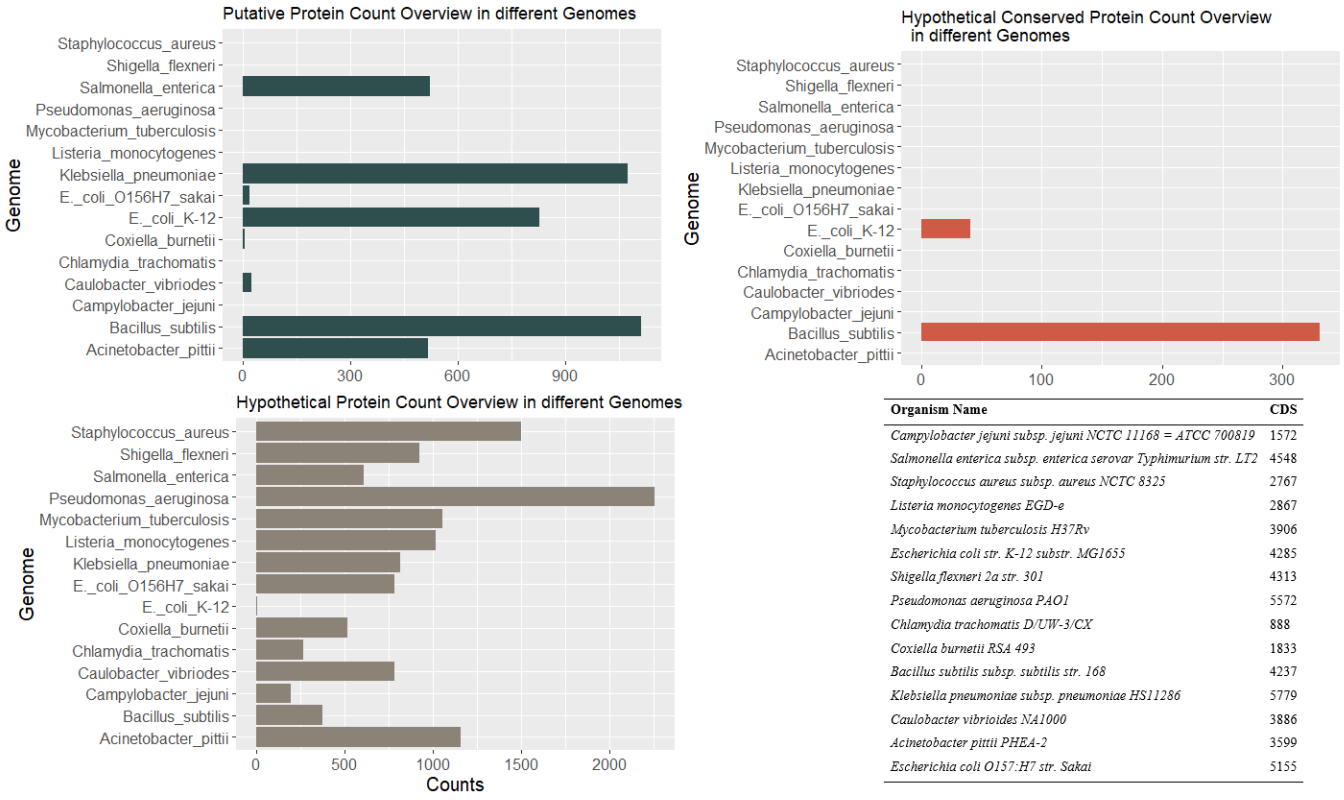
3.4 Chi square test

The chi square test was run on the *E. coli O157:H7 str. Sakai* K-mer dataset. The test was run between each feature in the dataset and the response (target) to determine if the association between the categorical variables reflects the association population-wise. It was also interesting to see if the feature selection using random forest and a statistical test found common important variables. The sklearn package in Python has a function chi2 that computes chi-squared stats between each non-negative feature and class (Michel, 2021).

4. Results

4.1 Overview of data

In this subsection the raw annotated data from the RefSeq database is presented. Here a mapping of the uncertain annotations is shown. The uncertain annotations are from most uncertain to least uncertain the proteins marked as “hypothetical”, “hypothetical conservative” and “putative”.



Organism Name	CDS
Campylobacter jejuni subsp. jejuni NCTC 11168 = ATCC 700819	1572
Salmonella enterica subsp. enterica serovar Typhimurium str. LT2	4548
Staphylococcus aureus subsp. aureus NCTC 8325	2767
Listeria monocytogenes EGD-e	2867
Mycobacterium tuberculosis H37Rv	3906
Escherichia coli str. K-12 substr. MG1655	4285
Shigella flexneri 2a str. 301	4313
Pseudomonas aeruginosa PAO1	5572
Chlamydia trachomatis D/UW-3/CX	888
Coxiella burnetii RSA 493	1833
Bacillus subtilis subsp. subtilis str. 168	4237
Klebsiella pneumoniae subsp. pneumoniae HS11286	5779
Caulobacter vibrioides NA1000	3886
Acinetobacter pittii PHEA-2	3599
Escherichia coli O157:H7 str. Sakai	5155

Figure 4.1.1 Count overview of different uncertain annotations present in the RefSeq annotated GFF files. The three figures are an overview of the three degrees of uncertain annotations present. A table with the total number of annotated genes in the different reference genomes are provided as a table in the bottom right corner.

The table 4.1.1 displays the count of different uncertain annotations per genome. The results were mapped using *ggplot2* version 3.3.2 into bar plots showing the number of different uncertain protein types for each reference sequence. The hypothetical protein count appears to be the most prominent uncertain annotation in all genomes except *E. coli K-12* and *B. subtilis*. For these genomes, the uncertain annotations that appear the most is the “putative protein” count.

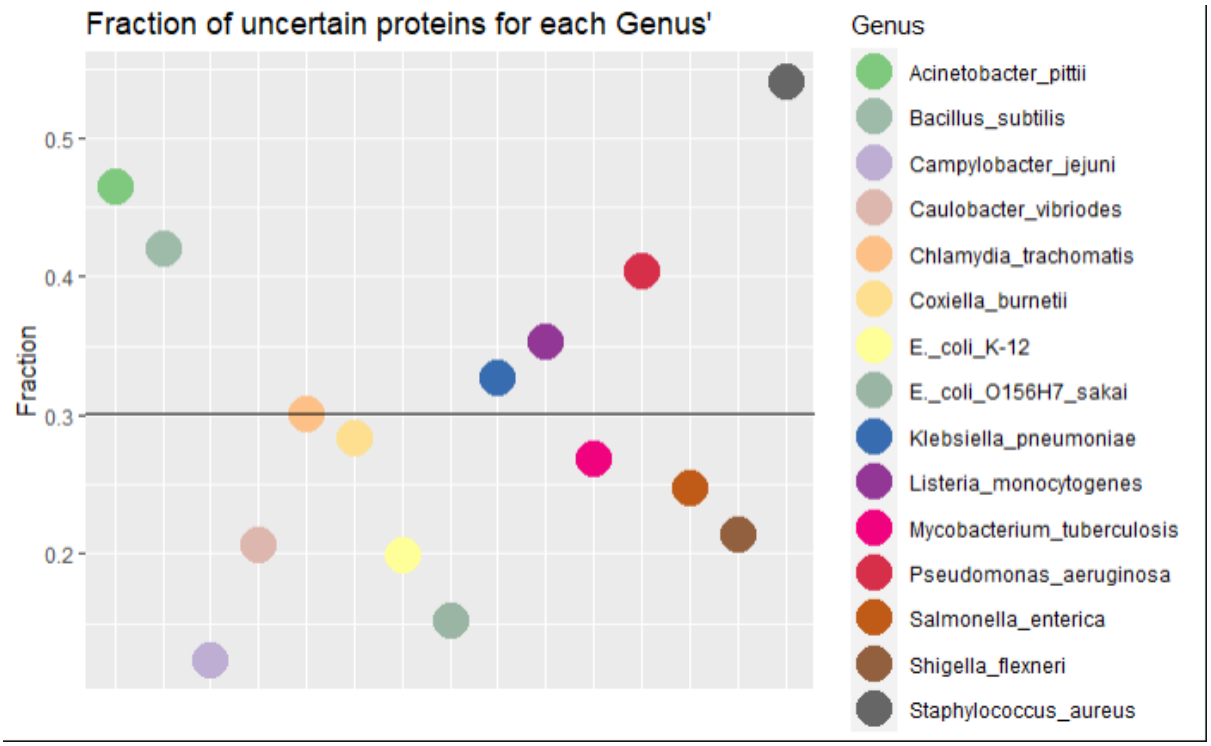


Figure 4.1.2 Dotplot showing the fraction of uncertain proteins for each genome. Black line exhibits mean percentage over all genomes.

The scatterplot, shown in figure 4.1.2, shows the fraction of uncertain proteins for each genus from the RefSeq fasta files. The results were mapped as a point graph with y being percentage (from 0 to 1), and the *x* value being different genera. The results give us the fraction of uncertain protein counts as well as the count and fraction of the individual uncertain proteins for each Genus. The mean of uncertain proteins is at around 0.3 with the minimum value being at 0.1 with *Bacillus subtilis* and maximum being at around 0.6 for *Staphylococcus aureus*.

There does not seem to be a trend for the fraction of uncertain proteins, there are also small differences species-wise as the *K-12* and *O156H7 sakai* subspecies of *E. coli* are not too far apart (0.1) fraction-wise.

4.2 Comparison with a gene prediction software

This segment presents an annotation tool's performance on the RefSeq sequence files. It was desirable to compare the RefSeq annotations with Prodigal predictions of the reference genomes. This comparison was made to illustrate a standard software's issue in locating the correct start-codon

In total Prodigal predicted 56 210 coding genes from the 15 different fasta files. The reference .gff file had a total of 55 538 observations leading to an overprediction of 672 observations. Out of these observations 53 505 observations matched the reference signature containing only end position, and 46 934 matched the reference signature containing both start and end position.

Table 4.2.1: Recall and precision between prodigal annotated genomes and known RefSeq genomes, panel A contains the results with SeqID, Strand type and end position (LORF), panel B with SeqID, Strand type, start and end position (ORF).

Genomes	Recall	Precision
Panel A: Signature for LORF		
Acinetobacter pittii	0.9814	0.9863
Bacillus subtilis	0.9600	0.9832
Campylobacter jejuni	0.9867	0.9397
Caulobacter vibriodes	0.9506	0.9919
Chlamydia trachomatis	0.9865	0.9766
Coxiella burnetii	0.8833	0.7867
Escherichia coli K-12	0.9578	0.9711
Escherichia coli O156H7 sakai	0.9853	0.9677
Klebsiella pneumoniae	0.9258	0.9808
Listeria monocytogenes	0.9927	0.9899
Mycobacterium tuberculosis	0.9703	0.9278
Pseudomonas aeruginosa	0.9957	0.9768
Salmonella enterica	0.9747	0.9579
Shigella flexneri	0.9573	0.8253
Staphylococcus aureus	0.9270	0.9749
Panel B: Signature for ORF		
Acinetobacter pittii	0.7774	0.7813
Bacillus subtilis	0.8688	0.8897
Campylobacter jejuni	0.9151	0.8715
Caulobacter vibriodes	0.7607	0.7938
Chlamydia trachomatis	0.8896	0.8807
Coxiella burnetii	0.6765	0.6025
Escherichia coli K-12	0.8913	0.9037
Escherichia coli O156H7 sakai	0.9679	0.9506
Klebsiella pneumoniae	0.8131	0.8614
Listeria monocytogenes	0.9279	0.9353
Mycobacterium tuberculosis	0.7496	0.7168
Pseudomonas aeruginosa	0.9142	0.8968
Salmonella enterica	0.8591	0.8442
Shigella flexneri	0.7714	0.6650
Staphylococcus aureus	0.8218	0.8643

The results for the prodigal comparison are shown in table 4.2.1. Panel A, with a signature that only contains SeqID, End position and Strand type, has a higher precision and recall overall when compared to panel B with end and start position included in the signature. For all genomes the precision and recall are higher when only end position is included in the signature.

The observations following the Prodigal annotation contained information about motifs. The software scans upstream information to see if any upstream sequences such as the Shine-Dalgarno are present. If they are found specified actions are taken, and the motif found is stored in the attribute's column of the output GFF files created. Results containing the frequency of a motif present can be seen in table 4.2.2. The comparison was made genome-wise, and the frequency shown is the total observations of motif *x*, for target *y*, genome *z*. The table shows the five possible Shine-Dalgarno RBS Motifs that Prodigal deems to have the highest score (Hyatt et al., 2010), as well as no RBS seat present.

Table 4.2.2: Shine-Dalgarno RBS motifs presence in upstream Prodigal annotated ORFs. The table shows the relative frequency of selected motifs present for four genomes of different targets. Panel A shows the relative frequency for target 1 (CDS) whereas panel B shows the relative frequency for target 0 (not CDS). NP means not present, as there are no observations.

Genome	**AGGAGG**	**AGGAG**	**AGGA**	**GGAGG**	**AGGA/GG AG/GAGG**	**None**
			Panel A: Gene			
B. subtilis	0.2035	0.1484	0.0457	0.1295	0.0212	0.0165
C. jejuni	0.0007	0.1674	0.3156	0.0069	NP	0.1287
C. vibriodes	NP	0.076	0.0575	0.0903	0.0182	0.1556
E. coli O157:H7 str. Sakai	0.0107	0.1400	0.1190	0.0588	0.0170	0.1084
			Panel B: not gene			
B. subtilis	0.0608	0.1139	0.0987	0.0810	0.0278	0.0861
C. jejuni	NP	0.1239	0.2743	0.0047	NP	0.2389
C. vibriodes	NP	0.0664	0.0556	0.0908	0.0203	0.1707
E. coli O157:H7 str. Sakai	0.0109	0.0652	NP	0.0217	0.0109	0.5

4.3 Chi square test

The Chi square test results are presented in table 4.3.1. The test was run on the K-mer data to test for enrichment in one class against the other.

Table 4.3.1 The twenty highest scores retrieved after running a chi square test for all K-mer features of the E. coli O157:H7 str. Sakai genome from 3-mer to 6-mer.

K-mer	Chi2 score	p-values
AGGA	1450.003	0.00E+00
GGAG	1287.984	4.62E-282
AGGAG	1252.086	2.92E-274
GAGG	1226.583	1.02E-268
GAG	1068.289	2.58E-234
AGG	1008.004	3.27E-221
TAA	998.0757	4.70E-219
AAGGA	988.911	4.62E-217
AAGGAG	674.4939	1.05E-148
TAAGG	608.5353	2.33E-134
GAGGT	606.1369	7.74E-134
GGAGA	604.307	1.94E-133
ATAA	582.1802	1.26E-128
GAGGA	550.4231	1.02E-121
GAGA	531.6311	1.25E-117
TAAG	526.679	1.49E-116
TAAGGA	525.6371	2.51E-116
GGA	524.6199	4.18E-116
AGGAGA	517.5854	1.42E-114
GGAGG	514.9453	5.32E-114

A chi square test was run for all K-mer data, and the twenty highest scores can be seen in table 4.3.1. This test was run between each feature and the response (target) to determine if the association between the categorical variables and target (gene or no gene) coincided. Based on the p-values presented, these 20 features are not independent from the response

variable and are thus deemed to be important factors for separating between gene and no gene in the upstream sequence data.

4.4 Modelling

In this subsection, the creation of training datasets and their attributes as well as the modelling performances are revealed. Information surrounding feature importance is also included in this segment.

4.4.1 The open reading frames

The reference table consisted of 55 538 observations, whereas the ORF table had 2 794 615 observations. After filtering the ORF table to only include the ORFs in the same longest reading frame as the reference table – the final dataset had a size of 1 107 520 observations. Out of these observations 55 234 were found to match the reference. Overall, this gives an average of 20 different reading frames per LORF.

Table 4.4.1 Overview of ORFs from reference and found from all genomes. This includes the false positives as well as true positives. Percentage of codings sequence loss (CDS loss) are also presented, as well as the number of uncertain proteins for each filtered length.

Type	**Number of observations**	**CDS loss**
Reference CDS'	55 538	
All ORFs that are CDS	55 234	0%
All ORFs that are not CDS	1 052 286	
Uncertain proteins	16 386	
CDS found with minimum ORF length 50	54 009	2,8%
ORFs that are not CDS minimum length 50	892 810	
Uncertain annotations of minimum length 50	16 376	
CDS found with minimum ORF length 90	50 226	9,6%
ORFs that are not CDS minimum length 90	760 890	
Uncertain annotations of minimum length 90	16 344	
CDS found with minimum ORF length 150	42 282	24%
ORFs that are not CDS minimum length 150	586 821	
Uncertain annotations of minimum length 150	15 459	

The loss of coding sequences when choosing an ORF length of 90 are at 9,6% as can be seen in table 4.4.1. By increasing the search space (lowering minimum ORF length) to 50, an increase of around 132 000 ORFs are observed. The minimum length of 90 bases was therefore selected for the final training datasets, to avoid more imbalanced datasets.

4.4.2 The Receiver Operating Characteristics

In total, seven models were trained on the *E. coli O1567:H7 sakai* genome. These models were the Balanced random forest classifier, the random forest classifier, logistic regression, gaussian naïve bayes, decision tree classifier, partial-least squares reduction followed by a linear discriminant analysis and the K-nearest neighbor classifier. As there are two training datasets (the K-mer dataset and the dummy-encoded dataset) the models were run on both datasets. The resulting ROC curves following the training can be seen in figure 4.4.1 and 4.4.2.

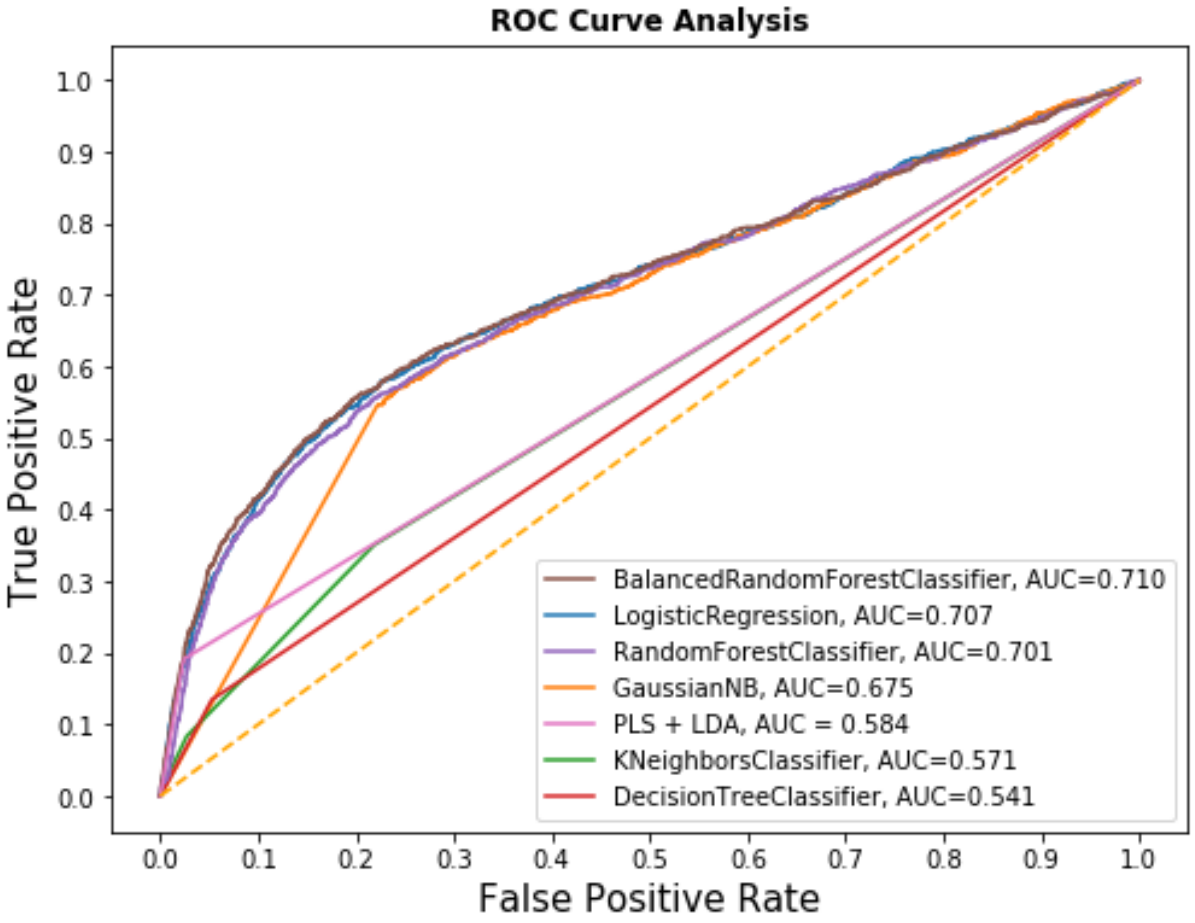


Figure 4.4.1 ROC curve for multiple classification models. The plot shows true positive rate versus false positive rate for the seven classifiers on the K-mer dataset for the E. coli O156:H7 sakai genome seen in table 3.3.4.1. AUC score is shown in the lower right quadrant of the figure.

The figure 4.4.1 shows the Receiver Operating Characteristics curve for the 7 different models trained in 2.4.3 on the *Escherichia coli O157:H7 sakai* genome. Based on the AUC scores, it appears the Balanced random forest classifier from imblearn had the most correct

classifications. The highest was at 0.71 and the lowest was the Decision trees classifiers at 0.541. There is little difference in AUC score between the top three classifiers, and the three may be fine classifiers for our training dataset.

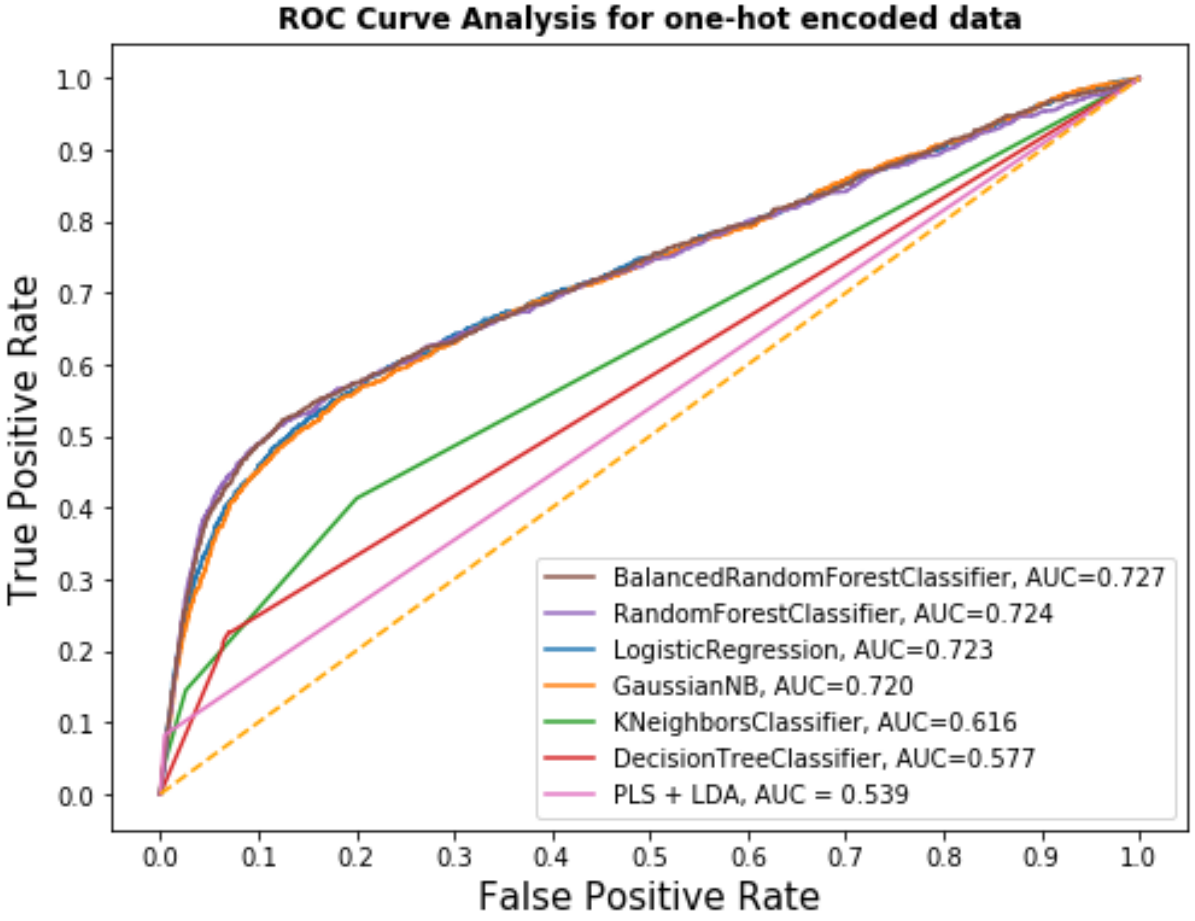


Figure 4.4.2 ROC curve for multiple classification models. The plot shows true positive rate versus false positive rate for the seven tuned classifiers on the one hot encoded dataset for the E. coli O156:H7 sakai genome. AUC score is shown in the lower right quadrant of the figure

The figure 4.4.2 shows the Receiver Operating Characteristics curve for the 7 different models trained in 2.4.3 on the *Escherichia coli O157:H7 sakai* genome. The dataset used was the sequential dataset that takes the order of nucleotides into account. Based on the AUC scores, it appears the Balanced random forest classifier from imblearn had the most correct classifications. The highest was at 0.727 and the lowest was the Decision trees classifiers at 0.577. There is little difference in AUC score between the top three classifiers, and all may be fine classifiers for our training dataset.

4.4.4 Four selected methods

This section presents the results from the four different classifiers trained on four representative genomes. The balanced random forest classifier as well as three common classifiers were chosen to better give an overview of the general training results. The classifiers were LDA+PLS, KNN and logistic regression. The models were tuned to best fit for each genome, and some selected metrics for four genomes can be seen in table 4.4.2 and 4.4.3. For the entire table see attachment 1 for the K-mer dataset and 2 for the sequential dataset. Training was performed with a hyperparameter tuning and a cross validation of 3. The resulting metrics are from the prediction of a validation test set.

Table 4.4.2 results from a hyperparameter tuned selection of four genomes. These values are selected according to the highest recall (panel A) and MCC score (panel B) from the table present in attachment 1. The classifier used is present in the column "Classifier" as is the minimum sum for a feature required before training (Reduction).

Genome	Precision	Recall	MCC	Classifier	Reduction
			Panel A		
B. subtilis	0.31	0.93	0.49	Balanced random forest	0.0
C. vibriodes	0.23	0.79	0.36	Balanced random forest	0.0
C. jejuni	0.32	0.78	0.43	Balanced random forest	1500.0
E. coli O156H7 sakai	0.13	0.6	0.19	Balanced random forest	0.0
			Panel B		
B. subtilis	0.68	0.48	0.55	logistic regression	500.0
C. jejuni	0.72	0.44	0.53	logistic regression	1000.0
C. vibriodes	0.6	0.27	0.37	logistic regression	500.0
E. coli O156H7 sakai	0.13	0.6	0.19	Balanced random forest	0.0

Table 4.4.2 shows the best results by Recall and MCC for the models trained using the K-mer dataset. Based on the results, the logistic regression scored highest in terms of MCC (panel

B), but the Balanced random forest classifier scored highest in terms of recall (panel A). The recall, when compared to the prodigal results are a little lower for all genomes except for *Bacillus subtilis* which has a higher recall rate than the prodigal ORF output (Table 4.1.1 panel B). For the most part, the reduction did not yield much difference score wise for the balanced random forest but proved to be an asset for the logistic regression classifier.

Table 4.4.3 results from a hyperparameter tuned selection of four genomes using the sequential dataset. These values are selected according to the highest recall and MCC score for each of the four genomes from the table present in attachment 2. The classifier used is present in the column "Classifier" as is the minimum sum for a feature required before training (Reduction).

Genome	Precision	Recall	MCC	Classifier	Reduction
			Panel A		
B. subtilis	0.48	0.92	0.64	Balanced random forest	0.0
C. vibriodes	0.31	0.9	0.48	Balanced random forest	0.0
C. jejuni	0.51	0.85	0.62	Balanced random forest	1000.0
E. coli O156H7 sakai	0.15	0.58	0.22	Balanced random forest	1000.0
			Panel B		
B. subtilis	0.83	0.68	0.73	logistic regression	0.0
C. jejuni	0.79	0.63	0.68	logistic regression	0.0
C. vibriodes	0.31	0.9	0.48	Balanced random forest	0.0
E. coli O156H7 sakai	0.15	0.58	0.22	Balanced random forest	1000.0

As for the K-mer dataset, logistic regression and the Balanced random forest classifier proved to be the best classification models for the sequential dataset. When sorting by highest recall for the four respective genomes (panel A), the Balanced Random Forest proved to give the highest scores, outdoing the prodigal scores for *Bacillus subtilis* and *Caulobacter vibriodes* in terms of recall for ORFs. The recall proved to be an asset for the random forest model for *C. jejuni* and *E. coli* with a reduction of 1000, which is a contrast to the K-mer dataset.

4.4.5 Balanced Random Forest classification

Because the Balanced Forest classifier had the highest AUC score in the ROC-curve analysis for the *E. coli O157:H7 str. Sakai* genome (Figures 4.4.1 and 4.4.2), this model was chosen for the remainder of the genomes.

The Balanced Random Forest model was run on all genomes in the dataset to get an overview of the variation between genomes. Table 4.4.4 and 4.4.5 shows the results following a balanced random forest classification on test data for respectively the K-mer and sequential dataset. Training was performed with a hyperparameter tuning and a cross validation of 3. The resulting metrics are from the prediction of a validation test set.

Table 4.4.4 results from a hyperparameter tuned Balanced random forest classifier for each genome in the reference genome database using K-mer data. The metrics Precision, Recall, MCC and F1 show their respective results. The column N trees show the optimal number of trees per genome.

Genome	**Precision**	**Recall**	**MCC**	**F1**	**N trees**
A. pittii	0.23	0.77	0.35	0.36	600
B. Subtilis	0.31	0.93	0.49	0.47	100
C. jejuni	0.31	0.77	0.43	0.44	800
C. vibriodes	0.23	0.79	0.36	0.35	300
C. trachomatis	0.16	0.72	0.24	0.26	100
C. burnetii	0.11	0.63	0.24	0.14	200
E. coli K-12	0.25	0.85	0.41	0.39	800
E. coli O156H7 sakai	0.13	0.59	0.18	0.21	500
K. pneumoniae	0.13	0.59	0.17	0.21	800
L. monocytogenes	0.40	0.91	0.55	0.55	500
M. tuberculosis	0.13	0.70	0.23	0.22	200
P. aeruginosa	0.23	0.84	0.39	0.37	400
S. enterica	0.13	0.61	0.18	0.21	900
S. flexneri	0.14	0.58	0.19	0.23	900
S. aureus	0.38	0.88	0.53	0.53	400

Table 4.4.4 shows the results from a hyperparameter tuned Balanced Random Forest classifier trained on the K-mer dataset. The genome with the highest performance in terms of MCC is *L. monocytogenes* with an optimal number of 500 trees and an MCC of 0.55.

Table 4.4.5 results from a hyperparameter tuned Balanced random forest classifier for each genome in the reference genome database using the sequential data. The metrics Precision, Recall, MCC and F1 show their respective results. The column N trees show the optimal number of trees per genome.

Genome	**Precision**	**Recall**	**MCC**	**F1**	**N trees**
A. pittii	0.3	0.8	0.43	0.43	700
B. subtilis	0.49	0.92	0.64	0.64	600
C. jejuni	0.51	0.85	0.62	0.63	700
C. vibriodes	0.31	0.9	0.48	0.46	600
C. trachomatis	0.21	0.82	0.34	0.34	600
C. burnetii	0.1	0.55	0.12	0.17	900
E. coli K-12	0.38	0.87	0.54	0.53	400
E. coli O156H7 sakai	0.15	0.58	0.21	0.24	900
K. pneumoniae	0.14	0.55	0.19	0.23	900
L. monocytogenes	0.59	0.92	0.71	0.72	800
M. tuberculosis	0.2	0.83	0.35	0.32	800
P. aeruginosa	0.35	0.92	0.54	0.51	800
S. enterica	0.15	0.59	0.21	0.24	800
S. flexneri	0.16	0.55	0.2	0.25	900
S. aureus	0.53	0.86	0.64	0.66	300

Table 4.4.5 shows the results from a hyperparameter tuned Balanced Random Forest classifier trained on the sequential dataset. The genome with the highest performance in terms of MCC is *L. monocytogenes* with an optimal number of trees at 800 and an MCC of 0.72.

4.4.6 Feature importances

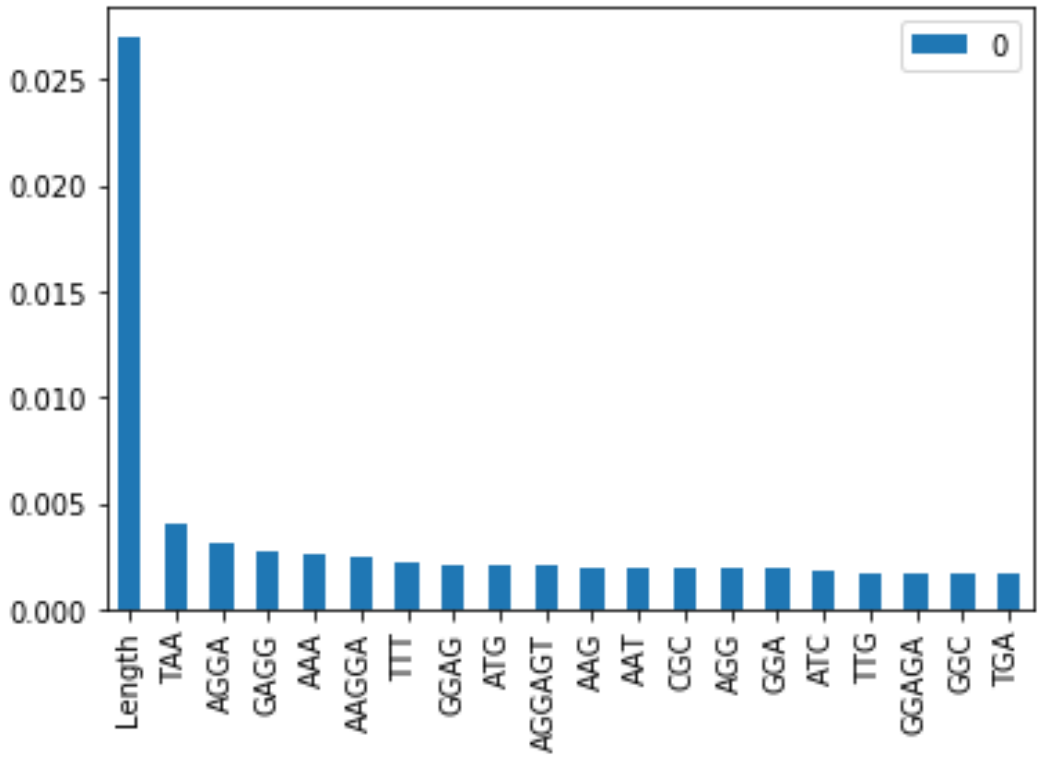


Figure 4.4.3 random forest feature importance. This barplot showcases the 20 most important features when constructing the random forest model for the genome of E. coli O156:H7 sakai. The importance is shown as fraction of importance.

Figure 4.4.3 highlights the random forest feature importance decisions. This feature importance displays what the random forest model considers as having the most information when classifying between gene (1) and no gene (0). The figure shows to 20 most important features in the training of the random forest model. The top variable appears to be the column length with a fraction of importance at 0.025, followed by the 3-mer TAA with an importance of around 0.004.

5. Discussion

5.1 Data

In this subsection the RefSeq data will be presented as well as certain aspects surrounding them. It is necessary to discuss the source of data as well as their unique attributes to understand the final classification results.

Figures 4.1.1 and 4.1.2 shows a count of the three different uncertain proteins for each genome. *C. jejuni* has the lowest fraction of uncertain proteins, indicating a nearly complete and accurate genome annotation. In figure 4.1.2 it appears that the fraction of uncertain proteins for *C. jejuni* is at 0,05 or 5%. In contrast the genome with the highest percentage of uncertain proteins is *S. aureus* at around 0,6 or 60%. It appears near all uncertain protein counts for *S. aureus* are hypothetical conserved proteins, which gives a little more credibility than just a hypothetical protein.

The annotation of the different genomes shows a wide variety in uncertain protein prediction and can be explained by individual characteristics of the genome, and the Genus. Different genera have different elements of variability. This variability can be explained by the different research focus' in the field. As some genera in the RefSeq database are model organisms, others are not. Due to some genera's status as model organisms there are often more experimental and manual annotation data available.

There also appears to be a difference between different strains. *E. coli K-12* and *O157:H7* are different in their count of protein. Where *K-12* has 20% count of uncertain protein, *O157:H7* has 15% (see figure 4.1.2). *K-12* has a high putative count (see figure 4.1.1) which gives more credibility than *O157:H7* with a higher hypothetical count. The difference between these is that the *K-12* strain is classified as a model organism and therefore has a high count of experimental data available in different databases. A quick search in the nucleotide database of GenBank yielded 6988 results for *K-12* and 643 results for *O157:H7*. *K-12* was also one of the first microorganisms targeted for genome sequencing (Perna, 2002), whereas the *E. coli O157:H7* is a strain associated with hemolytic-uremic syndrome and is not as commonly used as a model organism (Ameer, 2021). This may give an explanation as to why most protein counts of *O157:H7* is purely hypothetical and give insight into the differences among strains.

The RefSeq annotations are, as stated in the introduction, the best manual annotations we have as of today. These manual annotations are continuously being worked on and may contain errors as well. In the past, all RefSeq genome assemblies were reannotated once every few years to ensure that the older genomes benefit from the improvements in PGAP (Li et al., 2021). The latest publication from the RefSeq project at NCBI talks about the culling of bad proteins and the shrinking of the homology search space. There is, in other words, a great possibility that some CDS marked as genes in the current training dataset may be wrongly annotated and that some observations might be removed in the future.

Another aspect of the RefSeq annotated genomes is that the localization of the start-codon may not be certain. As stated, some software struggle with start site prediction in protein-coding genes. The PGAP uses GeneMarkS-2+ for start-site recognition. This software has a self-stated error rate of 4.4% compared to Prodigal which has 6.2% (Lomsadze et al., 2018). These self-stated errors can be thought of more as a minimum than a mean. Improvements on the error rate for GeneMarkS-2+ will lead to a more error-free training dataset in the future and may result in changes to the current start-positions of annotated genes.

5.2 Comparison with a gene prediction software

This section will discuss the Prodigal result. The Prodigal results presented the initial problem statement regarding annotation errors. By analyzing the annotation from the Prodigal result more insight into an annotation software's function is accomplished. The Prodigal annotation precision and recall creates a baseline for the final machine learning model.

Prodigal predicted 53 505 observations that matched the signature LORF (that contains only end position) and 46 934 matched the signature ORF (containing start and end). This gives an accuracy over the positive class to respectively 0,963 and 0,845. The accuracy is in other words reduced quite substantially when trying to estimate the correct start position of a coding sequence.

In table 4.2.1 the drop in recall and precision can be seen for all genomes when going from the signature for LORF (panel A) to signature for ORF (panel B). Indicating that the issue does not only lie in a few selected genomes. The results vary amongst the different genomes. The lowest precision and recall were found for the genome: *Coxiella burnetii* for both panel A

and B. The highest precision and recall were found for *E. coli O156H7 sakai* in 4.2.1 for the ORF signature (panel B). In panel A the genome with the highest precision was found to be *Pseudomonas aeruginosa,* and for recall *Listeria monocytogenes.*

There are two more motifs not present in panel B compared to panel A of table 4.2.2. Table 4.2.2 only contains ribosomal motif frequencies for four selected genomes, namely *E. coli O157:H7 sakai, B. subtilis, C. jejuni* and *C. vibriodes*. Panel B contains the relative frequency for motifs per genome present in non-coding ORFs whereas panel A contains the relative frequency of motif per genome for coding sequences. For the genomes, there is a lack of motif for non-coding ORFs. For *E. coli O157:H7* 50% of observations in panel B does not contain an upstream motif recognized by Prodigal. In contrast only 10.8% of observations for actual genes do not contain a motif for the same genome. There are, in general a higher relative percentage of motifs in panel A compared to panel B, apart from the three motifs AGGA/GGAG/GAGG.

The prodigal paper states to have distance-based scores when searching for ribosomal binding sites (Hyatt et al., 2010). When comparing the prodigal article with the output-data from prodigal, no distance-based score was discovered. It would have been more informative to retrieve a comparison of the motif AGGAGG with the length of spacer between the 16sRNA binding site and the start-codon as that was their top-scorings motifs, but no such information was found.

5.3 Classification based on upstream sequences

The machine learning classification model presented here is not as complex as any given annotation software. Given the time constraint on this thesis only a small part of what would be considered annotation has been investigated. Namely, the localization of the start position of genes based on upstream sequence data. This subsection presents points around the classification results.

The Balanced random forest classifier was deemed to be the best for both our datasets (K-mer an sequential) for the *E. coli sakai* genome as seen in Figure 4.4.1 and 4.4.2. However, there are multiple classification models that had an AUC score very similar to the balanced random forest one. Amongst them was the logistic regression model with an AUC of exactly 0.03

below the balanced random forest model for the K-mer dataset and 0.04 below for the sequential one. The ROC curves show more than which model has a better fit, it can also give insight into false positive and true positive rate. For an annotation process, an ideal step given the number of observations is to have as little false positives as possible. In other words, a conservative model is desired if this were the entire annotation model.

The need for a conservative model is dependent on its purpose. Usually, an annotation process consists of many models divided into a pipeline. The first model usually weeds out some false positives, but still manage to hold onto as many true positives as possible (a high recall score). Ultimately the goal is to retain only the true positives. When moving down the pipeline the need for a higher precision grows, as one needs to filter away the false positives. To retrieve a higher precision, the threshold can be raised to only retain what the model deems as very certain true positives. This balance can be hard to achieve, however, as the true labels are not always known. Especially when new data is involved.

The ROC curves for the Balanced random forest classifier are quite similar in terms of cutoff for the two different datasets, with the sequential proving to be more conservative in their classification of genes. The respective ROC curves seen in Figures 4.4.1 and 4.4.2 show the tradeoffs in false positive and false negative rates for different cutoffs in all the classification models. When looking at the curves for the balanced random forest, the optimal cutoff was observed at 0.6 positive rate and 0.2 false positive rate for Figure 4.1.1 (the K-mer data). For the sequential data, the optimal cutoff for the Balanced random forest classifier was at 0.52 true positive rate and around 0.13 for the false positive rate. For the sequential data this means they classify a positive observation as positive 52% of the time and a negative as positive 13% of the time. In general, this means both datasets produce similar results when used in training. In fact, there is a 71% chance that the random forest model produced from the K-mer dataset will be able to distinguish between a coding and a non-coding ORF, whereas the sequential has a 72,7% chance. The classification metrics presented here also coincide with the results for *E. coli sakai* in Tables 4.4.3 and 4.4.5.

Table 4.4.4 and 4.4.5 show metrics for respectively the K-mer and sequential dataset run for each genome using a Balanced random forest classifier with tuning. As stated in 5.1 the different strains of *E. coli* differ in number of sequences available. Thus, some annotations present in the *O157 sakai* genome may not be as plausible. The *K-12* strain performs better

than the *sakai* strain in the final modelling for both the sequential and the K-mer dataset, on all metrics. On the other hand, for the Prodigal ORF comparison, the *sakai* data had the highest precision and recall. Classification based purely on upstream data may prove to be a better fit for the *K-12* genome than for the *sakai*, since the Prodigal results are based on more than only the upstream sequence of each start codon.

The models trained and presented in Tables 4.4.4 and 4.4.5 all had a big false positive rate. The tables showed the results from a Balanced random forest classifier for all genomes trained on K-mer and sequential data. The precision metric returned yielded a minimum value of 0.1 and 0.11 for *C. burnetii* for the K-mer and sequential datasets respectively. The maximum precision was 0.40 and 0.59 both for *L. monocytogenes*. The results for *L. monocytogenes* are highly uncertain given that the genome has a fraction of uncertain protein count at 0.35, it is in other words not possible to decide if the score is close to the truth. In general, a low precision is an indication of the machine learning model easily overpredicting not-genes as genes. In an annotation process it is more desirable to have many false positives than negatives, as software usually have many more steps in the annotation process than just a single model. The issue with a large false positive rate, may, in other words not a problem if the model trained in this thesis were the first of many models.

In Table 4.4.2B and 4.4.3B the classifiers that gave the highest MCC score by genome was a mix between logistic regression and Balanced random forest classifier. For the genomes that had the logistic regression classifier as the highest MCC score, their other metrics are quite balanced. The genomes in question are *B. subtilis*, *C. jejuni* and *C. vibriodes* for the K-mer data and *B. subtilis* and *C. jejuni* for the sequential data. The precision for the genomes is much higher than for the Balanced random forest classifier, indicating a conservative model. Moreover, the logistic regression models had much better results for the sequential dataset and with no reduction of observations (see table 4.4.3 panel B), in contrast to the K-mer dataset that performed best when the minimum sum needed in a feature was at around 500 (table 4.4.2 panel B).

The sequential data proved to be faster and give better results, however patterns were easier seen with the K-mer dataset and may prove to be more valuable-information wise. The sequential data contains information about the order of bases upstream of a given ORF. When looking at sequences it is often more desirable to look at subsequences than positions. The

reasoning behind this is that it is easier for an individual to retrieve the context from a given K-mer rather than the positioning of the bases. Further on, most of already existing theory has evolved around substrings of genomes, which is what a K-mer fundamentally is.

The MCC score is the preferred metric for the data given the big imbalance in both datasets and should therefore be the deciding metric for the "best" classifier tested. When looking at the relative MCC scores for the four selected genomes for each dataset (see Table 4.4.2 and 4.4.3), the dataset with the highest MCC score was the sequential one. Table 4.4.4 panel B had two separate models that performed well, the Balanced random forest classifier for *E. coli sakai* and *C. vibriodes* and the logistic regression model for *C. jejuni* and *B. subtilis*. This division of classifier may initially seem like a stalemate, however given the preferences for annotation already stated in this subchapter, the model with the combination of high precision and high MCC score should be stated as the better model. A general conclusion can be made that a logistic regression model with a sequential dataset performed best out of the classifiers and datasets tested. This statement is made with regards to the method tested in this thesis with the selected parameters and processing.

When comparing the results from both datasets with prodigal all metrics fall below the baseline. However, in terms of recall the average recall score for Prodigal in terms of ORF signature is 0.83. For the Balanced random forest classifier run on all genomes the average is for K-mer and sequential data respectively at 0.74 and 0.77. This means a difference of 0.09 and 0.06 in terms of recall. Seeing as Prodigal takes more than just upstream data into account, the results are quite good. Considering precision score, however, the prodigal data yielded a precision of 0.82, whereas the K-mer model had a mean of 0.22 and the sequential 0.3. The sequential dataset proves to yield a higher precision indicating that the order of sequences yields more distinguishable information than the count of substrings. However, this difference is negligible as the precision proves to be quite low, specifically when considering the uncertainness in target labelling. In summary, the upstream sequence data alone is not enough to discern between coding and non-coding ORFs.

5.4 Motif importances in K-mer data

The K-mer data consists of several substrings, each substring being a feature in the dataset. The features present in the K-mer dataset may or may not be important for discerning between coding and non-coding ORFs. This subsection will discuss the results from both the Chi-squared test and the random forest feature permutation.

Both the feature selection method and the chi square method attempted (Table 4.3.1 and Figure 4.4.3) overlapped each other. The most important feature was deemed to be the 4mer AGGA for the chi-squared test and the feature length for random forest feature permutation. Continuous variables are not possible to assess in a chi-squared test and thus we assume TAA to be the most important categorical feature for comparison. The purine rich sequences (consisting of A and G) appear to be the most prominent K-mers from the feature importance visualization, as well as the Chi-square test. This corresponds with the upstream ribosomal binding site regions for CDS'. However, the feature selection does not tell us what target variable has the most or least amount of purine rich sequences. They only state that those K-mers are an important variable for differing between the targets (a test of homogeneity). The chi-square did show enriched K-mers, but that only means they appear a little more than random. It does not mean there even is a signal one can use for classification.

The TAA (a stop codon) is prominent in the classification of gene for the Random Forest Feature Permutation, along with the motifs typically found in the Ribosomal Binding site. The 3-mer may be present in operon-gene-clusters, although the importance does not seem to account for much (around 0.4%) when compared to length of an ORF (around 2.5%). Another possibility for the TAA 3-mer is an importance as part of the promoter region. The sigma70 holoenzyme attaches itself to the sequence seen in (1). The latter part being around 10 bases upstream of a CDS and containing TAA. Table 4.3.1 shows significant partial sequences from the sigma70 promoter sequence (ATAA), which coincides with underlying theory.

5.5 Limitations and further research

There are many directions to take when it comes to genome annotation. In this thesis the focus was to apply theoretical knowledge concerning the start position of a gene and see if it coincided in general for a machine learning application. Due to constraints only one aspect surrounding the prokaryotic genome has been investigated, namely aspects surrounding the upstream sequence data. In this section some limitations and suggestions are made for further research.

The first, most obvious limitation is the existence of operons. Operons prove to be difficult as they consist of gene clusters with one promoter upstream of the first gene in the cluster. This limits the upstream machine learning classification to genes with little to no operons, which is not a case in the prokaryotic genome. In fact, there is a possibility that many of the false negative classifications are genes in an operon cluster. A possible way to counterbalance these false negatives is to account for distance between LORFs in the final training dataset. A small distance meaning a higher chance of being in a cluster. Another possibility would be to make an in-depth analysis of the operon's functions and see if there is a general pattern one can use to classify such operons.

Ideally a generalized model trained on all coding sequences present in different genomes would have been a better fit for a future annotation model. The modelling results presented in this thesis are made genome wise to account for the variation between them. However, not all sequences have known species. When sequencing a metagenome, the species information is lost. This information loss would make the models redundant. A model that could be able to classify sequences without species definition would, in a metagenomic case, be beneficial. It would be advantageous to map the difference in performance between a generalized model trained on all 15 genomes against a model trained on only one genome.

Another limitation is the high dependency on the RefSeq annotations for the training datasets. As mentioned in the Section 5.1, the RefSeq annotation pipeline is still a work in progress and the data that the thesis surrounds itself around is not error-free. The fact that the annotation data contains errors can have led to false conclusions for the machine learning models. A wrong target label may lead to the supervised model making false assumptions, and if there are enough, lead to an entirely useless model. For instance the LDA classifier is shown to be

typically inconsistent in the present of label noise unless the prior probabilities of each class are equal (Cannings, 2019).

Some label noise could have been removed by eliminating the uncertain protein annotations. This would have led to a total removal of 16 344 observations of the positive class (see table 4.4.1). This corresponds to a loss of 29,4% coding sequences. For the dataset with an ORF filter length of 90, the mean number of ORFs per LORF is 16. By removing the 16 344 observations an average of 263 944 non-coding observations would have been removed as well. This would have resulted in a dataset of 33 882 coding ORFs and 628 866 non-coding ORFs. The relation would have been the same, but the information present in the sequences may have been less noisy and more informative.

The removal of LORFs that do not match any LORFs in the RefSeq annotation data may have left out some true positive variation. Some may consider these “alternative LORFs” to be the truest of negative classes in a binary classification, as no open reading frame is said to be a coding sequence. In this thesis the LORFs that did not match the RefSeq LORFs were filtered away, leaving behind only the LORFs already present in the reference data. These true negatives may have contributed by explaining some variance in the dataset, but it is not certain. The initial problem statement regarding inclusion of the alternative LORFs was the balancing of skewness. The balanced random forest classifier is an ideal classifier in that sense as it randomly downsamples the majority class to be of same size as the minority and can counter this original problem. An idea is to train an unfiltered ORF dataset with the dataset presented in this thesis to see if the model can discern better between coding ORFs and non-coding ORFs. ORFs here being all the alternative start and end positions present in a genome.

Small proteins have been left out from the model training. By lowering the minimum length of an ORF the false positive rate increases rapidly at the cost of some information loss. In Table 4.2.1 the number of false positives was found to be at 990 000 whereas the true positives were found to be 50 226. The total true positive rate when comparing this to the reference was at 55 538, meaning a loss of 5312 observations of the positive class in the training dataset. These small proteins are typically hidden or excluded in genome annotations due to the large number of false positive prediction that occurs with an increased search space.

An ideal expansion to the training dataset would be to introduce small genes shorter than 90 bp. Smaller protein products are being recognized and can encode functional polypeptides or act as cis-translational regulators (Khitun et al., 2019). These small open reading frames (smORFs) have been overlooked in this thesis due to the issue of an imbalanced dataset in the training data. It is, however, a possibility to investigate the mapping of mRNA data to the genome to include these smORFs in the dataset (Weaver et al., 2019).

An idea in terms of future work would be to implement the model presented here as part of a pipeline. Given the high recall and false positives present it would be ideal to utilize the upstream sequence model paired with a Balanced random forest classifier as the first initial step of an annotation process. Ideally, the model could lower its threshold to become more liberal in its classification. Increasing the search space and making sure actual genes are moved further down the pipeline while false positives are slowly chipped away using more conservative methods from new datasets.

6. Conclusion

The initial aim of the thesis was to *"investigate if sequences upstream of a start codon in an ORF is informative enough to discern between coding and non-coding ORFs"*. Here two datasets with different attributes have been created, namely a K-mer and a sequential dataset. These datasets show many similarities in terms of results, and they both performed worse than Prodigal, our standard annotation software. The reasoning behind this is quite simple; Prodigal looks for more than just upstream sequences. As far as upstream sequences are concerned, the models managed to pull all the information available from both datasets, with very limited value. However, there is still much more information surrounding genes to base annotation around.

Experimentally, a general understanding in biology has been uncovered. However, evolution creates a larger scope of possibilities that may not always be as easy to model for the current data. Based on the results, there appears to be a pattern pointing to specific motifs in the dataset, but the classification results only manage to scrape the surface. An ideal step forward is to expand into a pipeline so that the complex false negative classifications may be explained.

Bibliography

Abril, J. F. C., S. (2019). Genome Annotation. In *Encyclopedia of Bioinformatics and Computational Biology*.

Ameer, M. W., A. Salen, P. (2021). Escherichia Coli (E Coli 0157 H7). *StatPearls*. https://www.ncbi.nlm.nih.gov/books/NBK507845/

Anders, J., Petruschke, H., Jehmlich, N., Haange, S. B., von Bergen, M., & Stadler, P. F. (2021). A workflow to identify novel proteins based on the direct mapping of peptide-spectrum-matches to genomic locations. *BMC Bioinformatics*, *22*(1), 277. https://doi.org/10.1186/s12859-021-04159-8

Arlot, S. C., A. (2010). A survey of cross-validation procedures

for model selection. *Statistics Surveys*, *4*, 40-79. https://doi.org/http://dx.doi.org/10.1214/09-SS054

Armstrong, J., Fiddes, I. T., Diekhans, M., & Paten, B. (2019). Whole-Genome Alignment and Comparative Annotation. *Annu Rev Anim Biosci*, *7*, 41-64. https://doi.org/10.1146/annurev-animal-020518-115005

Barker, M. R., W. . (2002). Partial least squares for discrimination. *Journal of chemometrics*, *17*, 166-173. https://doi.org/10.1002/cem.785

Bergstra, J. B., Y. (2010). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*, 281-305.

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*, 5-32. https://doi.org/https://doi.org/10.1023/A:1010933404324

Brenner, S. M., J. (2001). *Encyclopedia of Genetics*. Elsevier Science Inc.

Britannica, T. (2018). *Operon*. Retrieved 17.04 from https://www.britannica.com/science/operon

Campbell, J. *Exploring Machine Learning : Introducing scikit-learn for ML in Python*. https://cornell-library.skillport.com/skillportfe/main.action?path=summary/BOOKS/130007

Cannings, T., Fan, Y. Samworth, R. (2019). Classification with imperfect training labels

https://doi.org/

https://doi.org/10.48550/arXiv.1805.11505

Cebrat, S., Dudek, M. R., Mackiewicz, P., Kowalczuk, M., & Fita, M. (1997). Asymmetry of coding versus noncoding strand in coding sequences of different genomes. *Microb Comp Genomics*, *2*(4), 259-268. https://doi.org/10.1089/omi.1.1997.2.259

Chicco, D. (2017). Ten quick tips for machine learning in computational biology. *BioData Min*, *10*, 35. https://doi.org/10.1186/s13040-017-0155-3

Chicco, D., Totsch, N., & Jurman, G. (2021). The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioData Min*, *14*(1), 13. https://doi.org/10.1186/s13040-021-00244-z

Cover, T. M. H., P E. (1968). The Condensed Nearest Neighbor RUle. *IEEE Transactions on Information Theory*, *14*(3), 515-516. https://doi.org/10.1109/TIT.1968.1054155

developers, s.-l. (2007-2022). *Permutation feature importance*. Retrieved 01.05 from https://scikit-learn.org/stable/_sources/modules/permutation_importance.rst.txt

Dong, Y., Li, C., Kim, K., Cui, L., & Liu, X. (2021). Genome annotation of disease-causing microorganisms. *Brief Bioinform*, *22*(2), 845-854. https://doi.org/10.1093/bib/bbab004

Eddy, S. R. (2004). What is a hidden Markov model? *Nat Biotechnol*, *22*(10), 1315-1316. https://doi.org/10.1038/nbt1004-1315

Errington, J., & Aart, L. T. V. (2020). Microbe Profile: Bacillus subtilis: model organism for cellular development, and industrial workhorse. *Microbiology (Reading)*, *166*(5), 425-427. https://doi.org/10.1099/mic.0.000922

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*, 861–874. https://doi.org/doi:10.1016/j.patrec.2005.10.010

Flach, P. W., S. (2002). Repairing Concavities in ROC Curves. https://www.ijcai.org/Proceedings/05/Papers/0652.pdf

Gabbay, D. M. W., J. (2005). *A Practical Logic of Cognitive Systems* (Vol. 2). Elsevier. https://doi.org/https://doi.org/10.1016/S1874-5075(05)80027-0.

Galperin, M. Y. (2001). Conserved 'hypothetical' proteins: new hints and new puzzles. *Comp Funct Genomics*, *2*(1), 14-18. https://doi.org/10.1002/cfg.66

Galperin, M. Y., Kristensen, D. M., Makarova, K. S., Wolf, Y. I., & Koonin, E. V. (2019). Microbial genome analysis: the COG approach. *Brief Bioinform*, *20*(4), 1063-1070. https://doi.org/10.1093/bib/bbx117

Haft, D. H., DiCuccio, M., Badretdin, A., Brover, V., Chetvernin, V., O'Neill, K., Li, W., Chitsaz, F., Derbyshire, M. K., Gonzales, N. R., Gwadz, M., Lu, F., Marchler, G. H., Song, J. S., Thanki, N., Yamashita, R. A., Zheng, C., Thibaud-Nissen, F., Geer, L. Y., . . . Pruitt, K. D. (2018). RefSeq: an update on prokaryotic genome annotation and curation. *Nucleic Acids Res*, *46*(D1), D851-D860. https://doi.org/10.1093/nar/gkx1068

Hanley, J. M., B. (1982). The Meaning and Use of the Area

under a Receiver Operating

Characteristic (ROC) Curve. *Radiology*, *143*, 29-36.

Hunter, J. D. (2007). Matplotlib is a 2D graphics package used for Python for

application development, interactive scripting, and publication-quality

image generation across user interfaces and operating systems. *Computing in Science & Engineering*, *9*, 90-95. https://doi.org/10.1109/MCSE.2007.55

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*, 119. https://doi.org/10.1186/1471-2105-11-119

Iriarte, A., Lamolle, G., & Musto, H. (2021). Codon Usage Bias: An Endless Tale. *J Mol Evol*, *89*(9-10), 589-593. https://doi.org/10.1007/s00239-021-10027-z

Khitun, A., Ness, T. J., & Slavoff, S. A. (2019). Small open reading frames and cellular stress responses. *Mol Omics*, *15*(2), 108-116. https://doi.org/10.1039/c8mo00283e

Kozak, M. (1999). Initiation of translation in prokaryotes and eukaryotes. *Gene*, *234*(2), 187-208. https://doi.org/10.1016/s0378-1119(99)00210-3

Land, M., Hauser, L., Jun, S. R., Nookaew, I., Leuze, M. R., Ahn, T. H., Karpinets, T., Lund, O., Kora, G., Wassenaar, T., Poudel, S., & Ussery, D. W. (2015). Insights from 20 years of bacterial genome sequencing. *Funct Integr Genomics*, *15*(2), 141-161. https://doi.org/10.1007/s10142-015-0433-4

Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., Devon, K., Dewar, K., Doyle, M., FitzHugh, W., Funke, R., Gage, D., Harris, K., Heaford, A., Howland, J., Kann, L., Lehoczky, J., LeVine, R., McEwan, P., . . . International Human Genome Sequencing, C. (2001). Initial sequencing and analysis of the human genome. *Nature*, *409*(6822), 860-921. https://doi.org/10.1038/35057062

Li, W., O'Neill, K. R., Haft, D. H., DiCuccio, M., Chetvernin, V., Badretdin, A., Coulouris, G., Chitsaz, F., Derbyshire, M. K., Durkin, A. S., Gonzales, N. R., Gwadz, M., Lanczycki, C. J., Song, J. S., Thanki, N., Wang, J., Yamashita, R. A., Yang, M., Zheng, C., . . . Thibaud-Nissen, F. (2021). RefSeq: expanding the Prokaryotic Genome Annotation Pipeline reach with protein family model curation. *Nucleic Acids Res*, *49*(D1), D1020-D1028. https://doi.org/10.1093/nar/gkaa1105

Liu, Y. R., W. (2007). PLS and dimension reduction for classification. *Computational statistics*, *22*, 189-208.

Lomsadze, A., Gemayel, K., Tang, S., & Borodovsky, M. (2018). Modeling leaderless transcription and atypical genes results in more accurate gene prediction in prokaryotes. *Genome Res*, *28*(7), 1079-1089. https://doi.org/10.1101/gr.230615.117

Martinez-Cano, D. J., Reyes-Prieto, M., Martinez-Romero, E., Partida-Martinez, L. P., Latorre, A., Moya, A., & Delaye, L. (2014). Evolution of small prokaryotic genomes. *Front Microbiol*, *5*, 742. https://doi.org/10.3389/fmicb.2014.00742

Mejia-Almonte, C., Busby, S. J. W., Wade, J. T., van Helden, J., Arkin, A. P., Stormo, G. D., Eilbeck, K., Palsson, B. O., Galagan, J. E., & Collado-Vides, J. (2020). Redefining fundamental concepts of transcription initiation in bacteria. *Nat Rev Genet*, *21*(11), 699-714. https://doi.org/10.1038/s41576-020-0254-8

Michel, V. T., B. Varoquaux, G. Gramfort, A. Duchesnay, E. Buitinck, L. Joly, A. (2021). *_univariate_selection.py*. In https://github.com/scikit-learn/scikit-learn/blob/baf828ca1/sklearn/feature_selection/_univariate_selection.py#L170

Minsky, M. (1961). Steps Toward Artificial Intelligence. *Proc. IRE*, *49*, 8.30.

Mir, K., Neuhaus, K., Scherer, S., Bossert, M., & Schober, S. (2012). Predicting statistical properties of open reading frames in bacterial genomes. *PLoS One*, *7*(9), e45103. https://doi.org/10.1371/journal.pone.0045103

Mistry, J., Chuguransky, S., Williams, L., Qureshi, M., Salazar, G. A., Sonnhammer, E. L. L., Tosatto, S. C. E., Paladin, L., Raj, S., Richardson, L. J., Finn, R. D., & Bateman, A. (2021). Pfam: The protein families database in 2021. *Nucleic Acids Res*, *49*(D1), D412-D419. https://doi.org/10.1093/nar/gkaa913

NCBI. (2021a). *NCBI Prokaryotic Genome Annotation Pipeline*. Retrieved 31.01 from https://www.ncbi.nlm.nih.gov/genome/annotation_prok/

NCBI. (2021b). *Prokaryotic RefSeq Genomes*. NCBI. Retrieved 31.01 from https://www.ncbi.nlm.nih.gov/refseq/about/prokaryotes/
NCBI. (2021c). *Prokaryotic RefSeq Genomes list*. Retrieved 31.01 from https://www.ncbi.nlm.nih.gov/genome/browse#!/prokaryotes/refseq_category:reference
NCBI. (2022). *GenBank and WGS Statistics*. Retrieved 05.05 from https://www.ncbi.nlm.nih.gov/genbank/statistics/
Oliveira, M. M., D Q. Ferrari, L I. Vasconcelos, A T R. (2004). Ribosome binding site recognition using neural networks. *Genetics and Molecular Biology*, *27*(4), 644-650. https://doi.org/ https://doi.org/10.1590/S1415-47572004000400028
Omotajo, D., Tate, T., Cho, H., & Choudhary, M. (2015). Distribution and diversity of ribosome binding sites in prokaryotic genomes. *BMC Genomics*, *16*, 604. https://doi.org/10.1186/s12864-015-1808-6
Palleja, A., Harrington, E. D., & Bork, P. (2008). Large gene overlaps in prokaryotic genomes: result of functional constraints or mispredictions? *BMC Genomics*, *9*, 335. https://doi.org/10.1186/1471-2164-9-335
Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, *2*, 559-572. https://doi.org/https://doi.org/10.1080/14786440109462720
Pearson, W. R. (2013). An introduction to sequence similarity ("homology") searching. *Curr Protoc Bioinformatics*, *Chapter 3*, Unit3 1. https://doi.org/10.1002/0471250953.bi0301s42
Pedregosa, F. V., G. Gramfort, A. Michel, V. Thirion, B. Grisel, O. Blondel, M. Prettenhofer, P. Weiss, R. Dubourg, V. Vanderplas, J. Passos, A. Cournapeau, D. Brucher, M. Perrot, M. Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825−2830.
Perna, T. G., J. Burland, V. Plunket III, G. (2002). *Virulence Mechanisms of a Versatile Pathogen*. Elsevier Inc.
Perry, S. C., & Beiko, R. G. (2010). Distinguishing microbial genome fragments based on their composition: evolutionary and comparative genomic perspectives. *Genome Biol Evol*, *2*, 117-131. https://doi.org/10.1093/gbe/evq004
R Development Core Team. (2010). *R: A language and environment for statistical computing*. In R Foundation for Statistical Computing. http://www.R-project.org
Raschka, S. M., V. (2019). *Python Machine Learning* (3 ed.). Packt Publishing Ltd.
Ruis-Perez, D. G., H. Madhivanan, P. Mathee, K. Narasimhan, G. (2020). So you think you can PLS-DA? *BMC Bioinformatics*, *21*. https://doi.org/https://doi.org/10.1186/s12859-019-3310-7
sklearn. (N.Y). *Decision Tree Classifier*. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier
Snipen, L. (2017). *mpda*. In https://github.com/larssnip/mpda
Snipen, L., & Liland, K. H. (2016). *Basic Biological Sequence Analysis*. In (Version 2.1.5) [Package].
Snipen, L., & Liland, K. H. (2017). *findOrfs*. https://github.com/larssnip/microseq/blob/master/R/orfs.R
Stein, L. (2006, 18.08.2020). *Generic Feature Format Version 3 (GFF3)*. Retrieved 27.03 from https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md
Tatusova, T., DiCuccio, M., Badretdin, A., Chetvernin, V., Nawrocki, E. P., Zaslavsky, L., Lomsadze, A., Pruitt, K. D., Borodovsky, M., & Ostell, J. (2016). NCBI prokaryotic genome annotation pipeline. *Nucleic Acids Res*, *44*(14), 6614-6624. https://doi.org/10.1093/nar/gkw569
ThermoFisher. (n.d). *Ribosomal Binding Site Sequence Requirements*. Retrieved 26.04 from https://www.thermofisher.com/no/en/home/references/ambion-tech-support/translation-systems/general-articles/ribosomal-binding-site-sequence-requirements.html
Vinje, H., Snipen, L., & Liland, K. H. (2016). *Methods for 16S based taxonomic classification of prokaryotes*. In (Version 1.2) [R package].
Volkenborn, K., Kuschmierz, L., Benz, N., Lenz, P., Knapp, A., & Jaeger, K. E. (2020). The length of ribosomal binding site spacer sequence controls the production yield for intracellular and secreted proteins by Bacillus subtilis. *Microb Cell Fact*, *19*(1), 154. https://doi.org/10.1186/s12934-020-01404-2
Watson, J. D. B., T. Stephen, B. Alexander, G. Michael, L. Richard, L. . (1965/2014). *Molecular Biology of the Gene* (7th ed.). Pearson.
Weaver, J., Mohammad, F., Buskirk, A. R., & Storz, G. (2019). Identifying Small Proteins by Ribosome Profiling with Stalled Initiation Complexes. *mBio*, *10*(2). https://doi.org/10.1128/mBio.02819-18
Webb, G. I. B., J R. Wang, Z. (2005). Not So Naive Bayes: Aggregating One-Dependence Estimators. *Machine Learning*, *58*, 5 - 24. https://doi.org/https://doi.org/10.1007%2Fs10994-005-4258-6
Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

Wold, S. S., M. Eriksson, L. (2001). PLS-regression: a basic tool of chemometrics. *Chemometrics and Intelligent Laboratory Systems*, *58*(2), 109-130. https://doi.org/https://doi.org/10.1016/S0169-7439(01)00155-1

Attachments

Attachment 1

Complete table with metrics and best parameters for the four selected genomes after running of four different classifiers with the K-mer dataset

Precision	Recall	F1	MCC	Genome	Classifier	Tuning	Reduction
0.31	0.93	0.46	0.49	*Bacillus subtilis*	imblearn	'classifier__n_estimators': 500	0.0
0.61	0.48	0.54	0.51	*Bacillus subtilis*	logistic regression	'classifier': LogisticRegression(C=0.01), 'classifier__C': 0.01	0.0
0.66	0.16	0.26	0.31	*Bacillus subtilis*	knn	'classifier': KNeighborsClassifier(n_neighbors=10, p=1), 'classifier__n_neighbors': 10, 'classifier__p': 1	0.0
0.66	0.16	0.26	0.31	*Bacillus subtilis*	lda+pls		0.0
0.31	0.93	0.47	0.49	*Bacillus subtilis*	imblearn	'classifier__n_estimators': 300	500.0
0.68	0.48	0.56	0.55	*Bacillus subtilis*	logistic regression	'classifier': LogisticRegression(C=0.01), 'classifier__C': 0.01	500.0
0.7	0.2	0.31	0.35	*Bacillus subtilis*	knn	'classifier': KNeighborsClassifier(n_neighbors=10, p=1), 'classifier__n_neighbors': 10, 'classifier__p': 1	500.0
0.7	0.2	0.31	0.35	*Bacillus subtilis*	lda+pls		500.0
0.31	0.92	0.46	0.49	*Bacillus subtilis*	imblearn	'classifier__n_estimators': 300	1000.0
0.68	0.48	0.56	0.55	*Bacillus subtilis*	logistic regression	'classifier': LogisticRegression(C=0.1), 'classifier__C': 0.1	1000.0
0.69	0.2	0.31	0.35	*Bacillus subtilis*	knn	'classifier': KNeighborsClassifier(n_neighbors=10, p=1), 'classifier__n_neighbors': 10, 'classifier__p': 1	1000.0
0.69	0.2	0.31	0.35	*Bacillus subtilis*	lda+pls		1000.0
0.3	0.93	0.46	0.48	*Bacillus subtilis*	imblearn	'classifier__n_estimators': 300	1500.0
0.68	0.47	0.55	0.54	*Bacillus subtilis*	logistic regression	'classifier': LogisticRegression(C=100.0), 'classifier__C': 100.0	1500.0
0.65	0.19	0.29	0.33	*Bacillus subtilis*	knn	'classifier': KNeighborsClassifier(n_neighbors=10, p=1), 'classifier__n_neighbors': 10, 'classifier__p': 1	1500.0
0.65	0.19	0.29	0.33	*Bacillus subtilis*	lda+pls		1500.0
0.32	0.77	0.45	0.43	*Campylobacter jejuni*	imblearn	'classifier__n_estimators': 300	0.0
0.59	0.42	0.49	0.46	*Campylobacter jejuni*	logistic regression	'classifier': LogisticRegression(C=0.01), 'classifier__C': 0.01	0.0
0.49	0.43	0.46	0.41	*Campylobacter jejuni*	knn	'classifier': KNeighborsClassifier(n_neighbors=10, p=1), 'classifier__n_neighbors': 10, 'classifier__p': 1	0.0

0.49	0.43	0.46	0.41	*Campylobacter jejuni*	lda+pls		0.0
0.31	0.76	0.44	0.42	*Campylobacter jejuni*	imblearn	‘classifier__n_estimators’: 900	500.0
0.73	0.41	0.52	0.52	*Campylobacter jejuni*	logistic regression	‘classifier’: LogisticRegression(C=0.01), ‘classifier__C’: 0.01	500.0
0.71	0.25	0.38	0.4	*Campylobacter jejuni*	knn	‘classifier’: KNeighborsClassifier(n_neighbors=10), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 2	500.0
0.71	0.25	0.38	0.4	*Campylobacter jejuni*	lda+pls		500.0
0.32	0.76	0.45	0.42	*Campylobacter jejuni*	imblearn	‘classifier__n_estimators’: 900	1000.0
0.72	0.44	0.55	0.53	*Campylobacter jejuni*	logistic regression	‘classifier’: LogisticRegression(), ‘classifier__C’: 1.0	1000.0
0.72	0.2	0.31	0.35	*Campylobacter jejuni*	knn	‘classifier’: KNeighborsClassifier(n_neighbors=10, p=1), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 1	1000.0
0.72	0.2	0.31	0.35	*Campylobacter jejuni*	lda+pls		1000.0
0.32	0.78	0.45	0.43	*Campylobacter jejuni*	imblearn	‘classifier__n_estimators’: 300	1500.0
0.71	0.43	0.54	0.52	*Campylobacter jejuni*	logistic regression	‘classifier’: LogisticRegression(), ‘classifier__C’: 1.0	1500.0
0.74	0.19	0.31	0.36	*Campylobacter jejuni*	knn	‘classifier’: KNeighborsClassifier(n_neighbors=10, p=1), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 1	1500.0
0.74	0.19	0.31	0.36	*Campylobacter jejuni*	lda+pls		1500.0
0.23	0.79	0.35	0.36	*Caulobacter vibriodes*	imblearn	‘classifier__n_estimators’: 500	0.0
0.48	0.3	0.37	0.35	*Caulobacter vibriodes*	logistic regression	‘classifier’: LogisticRegression(C=0.01), ‘classifier__C’: 0.01	0.0
0.45	0.03	0.05	0.1	*Caulobacter vibriodes*	knn	‘classifier’: KNeighborsClassifier(), ‘classifier__n_neighbors’: 5, ‘classifier__p’: 2	0.0
0.45	0.03	0.05	0.1	*Caulobacter vibriodes*	lda+pls		0.0
0.22	0.78	0.34	0.34	*Caulobacter vibriodes*	imblearn	‘classifier__n_estimators’: 300	500.0
0.6	0.27	0.37	0.37	*Caulobacter vibriodes*	logistic regression	‘classifier’: LogisticRegression(C=0.01), ‘classifier__C’: 0.01	500.0
0.61	0.06	0.12	0.19	*Caulobacter vibriodes*	knn	‘classifier’: KNeighborsClassifier(n_neighbors=10, p=1), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 1	500.0
0.61	0.06	0.12	0.19	*Caulobacter vibriodes*	lda+pls		500.0
0.21	0.78	0.34	0.34	*Caulobacter vibriodes*	imblearn	‘classifier__n_estimators’: 500	1000.0
0.6	0.25	0.36	0.37	*Caulobacter vibriodes*	logistic regression	‘classifier’: LogisticRegression(C=0.01), ‘classifier__C’: 0.01	1000.0
0.6	0.09	0.16	0.22	*Caulobacter vibriodes*	knn	‘classifier’: KNeighborsClassifier(n_neighbors=10, p=1), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 1	1000.0
0.6	0.09	0.16	0.22	*Caulobacter vibriodes*	lda+pls		1000.0

0.21	0.77	0.33	0.34	*Caulobacter vibriodes*	imblearn	‘classifier__n_estimators’: 300	1500.0
0.63	0.23	0.34	0.36	*Caulobacter vibriodes*	logistic regression	‘classifier’: LogisticRegression(C=0.01), ‘classifier__C’: 0.01	1500.0
0.62	0.07	0.12	0.19	*Caulobacter vibriodes*	knn	‘classifier’: KNeighborsClassifier(n_neighbors=10), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 2	1500.0
0.62	0.07	0.12	0.19	*Caulobacter vibriodes*	lda+pls		1500.0
0.13	0.6	0.22	0.19	*E.coli O156H7 sakai*	imblearn	‘classifier__n_estimators’: 900	0.0
0.34	0.08	0.13	0.14	*E.coli O156H7 sakai*	logistic regression	‘classifier’: LogisticRegression(C=0.01), ‘classifier__C’: 0.01	0.0
0.33	0.0	0.0	0.01	*E.coli O156H7 sakai*	knn	‘classifier’: KNeighborsClassifier(n_neighbors=10, p=1), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 1	0.0
0.33	0.0	0.0	0.01	*E.coli O156H7 sakai*	lda+pls		0.0
0.13	0.59	0.21	0.18	*E.coli O156H7 sakai*	imblearn	‘classifier__n_estimators’: 600	500.0
0.4	0.05	0.09	0.13	*E.coli O156H7 sakai*	logistic regression	‘classifier’: LogisticRegression(C=0.01), ‘classifier__C’: 0.01	500.0
0.45	0.0	0.01	0.04	*E.coli O156H7 sakai*	knn	‘classifier’: KNeighborsClassifier(n_neighbors=10), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 2	500.0
0.45	0.0	0.01	0.04	*E.coli O156H7 sakai*	lda+pls	‘classifier’: KNeighborsClassifier(n_neighbors=10), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 2	500.0
0.13	0.59	0.22	0.19	*E.coli O156H7 sakai*	imblearn	‘classifier__n_estimators’: 600	1000.0
0.45	0.04	0.08	0.13	*E.coli O156H7 sakai*	logistic regression	‘classifier’: LogisticRegression(C=0.01), ‘classifier__C’: 0.01	1000.0
0.29	0.0	0.01	0.03	*E.coli O156H7 sakai*	knn	‘classifier’: KNeighborsClassifier(n_neighbors=10, p=1), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 1	1000.0
0.29	0.0	0.01	0.03	*E.coli O156H7 sakai*	lda+pls		1000.0
0.13	0.58	0.22	0.18	*E.coli O156H7 sakai*	imblearn	‘classifier__n_estimators’: 900	1500.0
0.48	0.05	0.09	0.14	*E.coli O156H7 sakai*	logistic regression	‘classifier’: LogisticRegression(C=0.1), ‘classifier__C’: 0.1	1500.0
0.29	0.0	0.01	0.03	*E.coli O156H7 sakai*	knn	‘classifier’: KNeighborsClassifier(n_neighbors=10, p=1), ‘classifier__n_neighbors’: 10, ‘classifier__p’: 1	1500.0
0.29	0.0	0.01	0.03	*E.coli O156H7 sakai*	lda+pls		1500.0

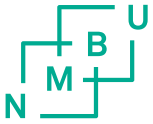
Attachment 2

Complete table with metrics and parameters after running of the four different classifiers on the four selected genomes with the sequential dataset

Precision	Recall	F1	MCC	Genome	Classifier	Tuning	Reduction
0.48	0.92	0.63	0.64	*Bacillus subtilis*	imblearn	'classifier__n_estimators': 800	0.0
0.48	0.92	0.63	0.64	*Bacillus subtilis*	imblearn	'classifier__n_estimators': 700	500.0
0.48	0.92	0.63	0.64	*Bacillus subtilis*	imblearn	'classifier__n_estimators': 800	1000.0
0.48	0.92	0.63	0.63	*Bacillus subtilis*	imblearn	'classifier__n_estimators': 700	1500.0
0.51	0.85	0.64	0.62	*Campylobacter jejuni*	imblearn	'classifier__n_estimators': 700	1000.0
0.5	0.84	0.63	0.61	*Campylobacter jejuni*	imblearn	'classifier__n_estimators': 600	0.0
0.5	0.84	0.63	0.61	*Campylobacter jejuni*	imblearn	'classifier__n_estimators': 500	500.0
0.5	0.84	0.62	0.61	*Campylobacter jejuni*	imblearn	'classifier__n_estimators': 500	1500.0
0.31	0.9	0.46	0.48	*Caulobacter vibriodes*	imblearn	'classifier__n_estimators': 600	0.0
0.31	0.9	0.46	0.48	*Caulobacter vibriodes*	imblearn	'classifier__n_estimators': 500	1000.0
0.31	0.89	0.46	0.48	*Caulobacter vibriodes*	imblearn	'classifier__n_estimators': 400	500.0
0.3	0.89	0.45	0.47	*Caulobacter vibriodes*	imblearn	'classifier__n_estimators': 500	1500.0
0.15	0.58	0.24	0.22	*E. coli O156H7_sakai*	imblearn	'classifier__n_estimators': 900	1000.0
0.15	0.58	0.24	0.21	*E. coli O156H7_sakai*	imblearn	'classifier__n_estimators': 800	1500.0
0.15	0.57	0.24	0.21	*E. coli O156H7_sakai*	imblearn	'classifier__n_estimators': 800	0.0
0.15	0.57	0.24	0.21	*E. coli O156H7_sakai*	imblearn	'classifier__n_estimators': 900	500.0
0.78	0.46	0.58	0.58	*Bacillus subtilis*	knn	'classifier': KNeighborsClassifier(p=1), 'classifier__n_neighbors': 5, 'classifier__p': 1	0.0
0.78	0.46	0.58	0.58	*Bacillus subtilis*	knn	'classifier': KNeighborsClassifier(p=1), 'classifier__n_neighbors': 5, 'classifier__p': 1	500.0
0.78	0.46	0.58	0.58	*Bacillus subtilis*	knn	'classifier': KNeighborsClassifier(p=1), 'classifier__n_neighbors': 5, 'classifier__p': 1	1000.0
0.78	0.46	0.58	0.58	*Bacillus subtilis*	knn	'classifier': KNeighborsClassifier(p=1), 'classifier__n_neighbors': 5, 'classifier__p': 1	1500.0
0.88	0.49	0.63	0.64	*Campylobacter jejuni*	knn	'classifier': KNeighborsClassifier(n_neighbors=10, p=1), 'classifier__n_neighbors': 10, 'classifier__p': 1	0.0
0.88	0.49	0.63	0.64	*Campylobacter jejuni*	knn	'classifier': KNeighborsClassifier(n_neighbors=10, p=1), 'classifier__n_neighbors': 10, 'classifier__p': 1	500.0
0.88	0.49	0.63	0.64	*Campylobacter jejuni*	knn	'classifier': KNeighborsClassifier(n_neighbors=10,	1000.0

						p=1), 'classifier__n_neighbors': 10, 'classifier__p': 1	
0.9	0.48	0.62	0.63	*Campylobacter jejuni*	knn	'classifier': KNeighborsClassifier(n_neighbors=10, p=1), 'classifier__n_neighbors': 10, 'classifier__p': 1	1500.0
0.73	0.1	0.17	0.25	*Caulobacter vibriodes*	knn	'classifier': KNeighborsClassifier(p=1), 'classifier__n_neighbors': 5, 'classifier__p': 1	0.0
0.73	0.1	0.17	0.25	*Caulobacter vibriodes*	knn	'classifier': KNeighborsClassifier(p=1), 'classifier__n_neighbors': 5, 'classifier__p': 1	500.0
0.73	0.1	0.17	0.25	*Caulobacter vibriodes*	knn	'classifier': KNeighborsClassifier(p=1), 'classifier__n_neighbors': 5, 'classifier__p': 1	1000.0
0.73	0.1	0.17	0.25	*Caulobacter vibriodes*	knn	'classifier': KNeighborsClassifier(p=1), 'classifier__n_neighbors': 5, 'classifier__p': 1	1500.0
0.35	0.01	0.01	0.04	*E. coli O156H7_sakai*	knn	'classifier': KNeighborsClassifier(n_neighbors=10), 'classifier__n_neighbors': 10, 'classifier__p': 2	0.0
0.35	0.01	0.01	0.04	*E. coli O156H7_sakai*	knn	'classifier': KNeighborsClassifier(n_neighbors=10), 'classifier__n_neighbors': 10, 'classifier__p': 2	500.0
0.35	0.01	0.01	0.04	*E. coli O156H7_sakai*	knn	'classifier': KNeighborsClassifier(n_neighbors=10), 'classifier__n_neighbors': 10, 'classifier__p': 2	1000.0
0.35	0.01	0.01	0.04	*E. coli O156H7_sakai*	knn	'classifier': KNeighborsClassifier(n_neighbors=10), 'classifier__n_neighbors': 10, 'classifier__p': 2	1500.0
0.74	0.7	0.72	0.7	*Bacillus subtilis*	lda+pls		0.0
0.74	0.7	0.72	0.7	*Bacillus subtilis*	lda+pls		500.0
0.74	0.7	0.72	0.7	*Bacillus subtilis*	lda+pls		1000.0
0.74	0.7	0.72	0.7	*Bacillus subtilis*	lda+pls		1500.0
0.76	0.68	0.72	0.70	*Campylobacter jejuni*	lda+pls		0.0
0.76	0.68	0.72	0.70	*Campylobacter jejuni*	lda+pls		500.0
0.76	0.68	0.72	0.70	*Campylobacter jejuni*	lda+pls		1000.0
0.76	0.68	0.72	0.70	*Campylobacter jejuni*	lda+pls		1500.0
0.51	0.53	0.52	0.48	*Caulobacter vibriodes*	lda+pls		0.0
0.51	0.53	0.52	0.48	*Caulobacter vibriodes*	lda+pls		500.0
0.51	0.53	0.52	0.48	*Caulobacter vibriodes*	lda+pls		1000.0
0.51	0.53	0.52	0.48	*Caulobacter vibriodes*	lda+pls		1500.0
0.54	0.08	0.14	0.19	*E. coli O156H7_sakai*	lda+pls		0.0

0.54	0.08	0.14	0.19	*E. coli O156H7_sakai*	lda+pls		500.0
0.54	0.08	0.14	0.19	*E. coli O156H7_sakai*	lda+pls		1000.0
0.54	0.08	0.14	0.19	*E. coli O156H7_sakai*	lda+pls		1500.0
0.83	0.68	0.75	0.73	*Bacillus subtilis*	logistic regression	'classifier': LogisticRegression(C=0.1), 'classifier__C': 0.1	0.0
0.83	0.68	0.75	0.73	*Bacillus subtilis*	logistic regression	'classifier': LogisticRegression(C=0.1), 'classifier__C': 0.1	500.0
0.83	0.68	0.75	0.73	*Bacillus subtilis*	logistic regression	'classifier': LogisticRegression(C=0.1), 'classifier__C': 0.1	1000.0
0.83	0.68	0.75	0.73	*Bacillus subtilis*	logistic regression	'classifier': LogisticRegression(C=0.1), 'classifier__C': 0.1	1500.0
0.79	0.63	0.7	0.68	*Campylobacter jejuni*	logistic regression	'classifier': LogisticRegression(), 'classifier__C': 1.0	0.0
0.79	0.63	0.7	0.68	*Campylobacter jejuni*	logistic regression	'classifier': LogisticRegression(), 'classifier__C': 1.0	500.0
0.79	0.63	0.7	0.68	*Campylobacter jejuni*	logistic regression	'classifier': LogisticRegression(), 'classifier__C': 1.0	1000.0
0.8	0.63	0.7	0.68	*Campylobacter jejuni*	logistic regression	'classifier': LogisticRegression(C=0.1), 'classifier__C': 0.1	1500.0
0.65	0.39	0.49	0.48	*Caulobacter vibriodes*	logistic regression	'classifier': LogisticRegression(), 'classifier__C': 1.0	0.0
0.65	0.39	0.49	0.48	*Caulobacter vibriodes*	logistic regression	'classifier': LogisticRegression(), 'classifier__C': 1.0	500.0
0.65	0.39	0.49	0.48	*Caulobacter vibriodes*	logistic regression	'classifier': LogisticRegression(), 'classifier__C': 1.0	1000.0
0.65	0.39	0.49	0.48	*Caulobacter vibriodes*	logistic regression	'classifier': LogisticRegression(), 'classifier__C': 1.0	1500.0
0.42	0.02	0.04	0.08	*E. coli O156H7_sakai*	logistic regression	'classifier': LogisticRegression(C=0.1), 'classifier__C': 0.1	0.0
0.42	0.02	0.04	0.08	*E. coli O156H7_sakai*	logistic regression	'classifier': LogisticRegression(C=0.1), 'classifier__C': 0.1	500.0
0.42	0.02	0.04	0.08	*E. coli O156H7_sakai*	logistic regression	'classifier': LogisticRegression(C=0.1), 'classifier__C': 0.1	1000.0
0.42	0.02	0.04	0.08	*E. coli O156H7_sakai*	logistic regression	'classifier': LogisticRegression(C=0.1), 'classifier__C': 0.1	1500.0



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

Postboks 5003
NO-1432 Ås
Norway