Einsatz von Machine Learning zur Evaluierung der Qualität von Next-Generation-Sequencing-Daten

Forschungsprojekt Teil A

Alexander Hinzer
Alexander.Hinzer@student.htw-berlin.de

Prüfer: Prof. Dr.-Ing. Piotr Wojciech Dabrowski

Studiengang: Angewandte Informatik (Master)
Fachbereich 4
Hochschule für Technik und Wirtschaft Berlin

Wintersemester 2021/22

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Zielstellung

In der vorliegenden Projektarbeit soll der Einsatz von Machine Learning (ML) zur Evaluierung der Qualität von Next-Generation-Sequencing-Daten untersucht werden. Dafür müssen geeignete Attribute dieser Sequenzierungsdaten herausgestellt und für die Implementierung verschiedener ML-Algorithmen verwendet werden. Die unterschiedlichen ML-Algorithmen sollen daraufhin in ihrer Fähigkeit, die Qualität der Sequenzierungsdaten zu evaluieren, beurteilt und miteinander verglichen werden. Diese Arbeit führt im ersten Teil in die Gewinnung und Verarbeitung von Next-Generation-Sequencing-Daten ein und erläutert daraufhin die Grundlagen der ML-Methoden, die für die Implementierung des Projekts notwendig sind. Abschließend werden die Ergebnisse des Projekts vorgestellt und diskutiert.

1.2 Motivation

Bevor Next-Generation-Sequencing-Daten einer genauen Analyse unterzogen werden, um biologische Schlussfolgerungen ziehen zu können, sollte zunächst eine Evaluierung ihrer Qualität durchgeführt werden. Damit soll sichergestellt werden, dass keine Fehler im Ausgangsmaterial oder dem Sequenzierungsgerät vorliegen und keine Störungen während des Sequenzierungsprozesses aufgetreten sind. Diese Qualitätskontrolle wird manuell durch Biologen ausgeführt und verursacht Arbeitsaufwand und Kosten, die durch eine Automatisierung dieses Prozesses reduziert werden können. Da die Entscheidungsfindung komplex ist und nicht durch klare Regeln definiert ist, könnten sich ML-Algorithmen dazu eignen, die Automatisierung umzusetzen.

1.3 Rahmenbedingungen

Für die vorliegende Untersuchung werden Daten von 184 Sequenzierungen verschiedener Bakterienproben vom Robert Koch-Institut in Berlin bereitgestellt. Die Qualitäten dieser Daten sind bereits von Biologen eingeschätzt und eignen sich damit zur Implementierung von ML-Algorithmen. Zur Umsetzung des Projekts soll Python, sowie die Software FastQC[1] eingesetzt werden.

2 Next-Generation-Sequencing-Daten

2.1 Illumina Sequenzierung

DNA-Sequenzierung ermöglicht es Nukleotidsequenzen von DNA-Molekülen zu bestimmen. Technologien der DNA-Sequenzierung haben seit ihrer Einführung einen großen Einfluss auf die biologische Forschung und eröffneten den neuen Wissenschaftszweig der Genomik. Die erste Generation dieser Technologien wurde mit der Didesoxymethode nach Sanger (1977) eingeleitet, war jedoch sehr arbeits- und kostenintensiv [2]. Durch stetige Weiterentwicklung und Einführung hochgradig parallelisierter Systeme können modernere Verfahren die Sequenzierung wesentlich effizienter und günstiger durchführen. Diese werden unter dem Begriff Next Generation Sequencing (NGS) zusammengefasst. Eine aktuell verbreitete und für die Erstellung der Daten des vorliegenden Projekts verwendete Technologie ist die Illumina-Sequenzierung nach dem Prinzip der Sequenzierung durch Brückensynthese. Der Ablauf von NGS-Verfahren lässt sich in vier Schritte einteilen, die am Beispiel der Illumina-Sequenzierung näher erläutert werden.

2.1.1 Bibliotheksvorbereitung

Um eine doppelsträngige DNA für die Sequenzierung vorzubereiten, wird eine Bibliothek von dieser erstellt. Dazu muss die DNA zunächst fragmentiert, also in kleinere doppelsträngige Stücke zerlegt werden. Dies kann zum Beispiel durch Ultraschallaufschluss umgesetzt werden, indem Ultraschallwellen kleine Gasblasen in der DNA-beinhaltenden Flüssigkeit erzeugen. Wenn diese Blasen kollabieren werden die Bindungen des DNA-Rückgrats aufgebrochen, wodurch einzelne Fragmente mit ungleichmäßigen Enden entstehen, also der obere und untere Strang unterschiedliche Längen aufweisen. Durch T4-DNA-Polymerase werden diese Überhänge mit den komplementären Basen[1] aufgefüllt. [3]

Für die Sequenzierung werden bekannte Sequenzabschnitte benötigt, um einen Startpunkt für die Primer bereitzustellen. Dafür werden Adapter, kurze doppelsträngige DNA Abschnitte mit bekannten Nukleotidsequenzen, hinzugefügt. Damit diese sich mit den Fragmenten verbinden können, bekommen die Fragmente durch eine Klenow-Polymerase an beiden Enden einen Adenin-Überhang. Somit können sich die Adapter, bei denen stets ein Thymin Überhang vorliegt, mittels DNA-Ligase mit den Enden der Fragmente verbinden. Alternativ kann durch den Einsatz von Tn5-Transposase die Fragmentierung auch gleichzeitig mit dem Hinzufügen der Adapter durchgeführt werden. Hierbei wird dem Tn5-Enzym zunächst der Adapter hinzugefügt. Wenn das Tn5 die doppelsträngige DNA schneidet, fügt es den Adapter direkt dem Fragment an. [3]

[1] In einem DNA-Doppelstrang bilden Basen über Wasserstoffbrücken Basenpaare. Dabei können sich nur jeweils Guanin mit Cytosin und Adenin mit Thymin verbinden, sie werden daher als komplementäre Basen bezeichnet.

2.1.2 Clusterbildung

Das Ziel der Clusterbildung ist jedes Fragment der Bibliothek in tausende identische Kopien zu klonen. Die Clusterbildung erfolgt auf einer Fließzelle. Die Fließzelle ist ein Glasobjektträger mit mikrofluidischen Kanälen, welche einen kontrollierten Durchfluss verschiedener Reagenzien ermöglichen. Außerdem ist die Oberfläche der Fließzelle mit, zu den Adaptern, komplementären Oligonukleotiden beschichtet. Wenn eine denaturisierte Bibliothek zu einer Fließzelle hinzugegeben wird, binden sich die Adapter [4].

Das Cluster wird durch Brücken-PCR erstellt, mit den Oligonukleotiden auf der Oberfläche als PCR-Primer. Dabei biegen sich die, an einem Ende verbundenen, Einzelstränge der Fragmente zu einem naheliegenden Oligonukleotid, sodass auch das zweite Ende mit der Oberfläche verbunden ist und der Strang einen Brückenbogen bildet [3].

Eine Polymerase synthetisiert daraufhin den reversen Strang, wodurch wieder ein doppelsträngiges Fragment vorliegt. Diese wird wieder denaturisiert, wodurch die Einzelstränge wieder aufklappen und sich mit einem weiteren Oligonukleotid auf der Oberfläche verbinden können. Dieser Amplifikationsvorgang, dargestellt in Abbildung 2.1, wird mehrere Male wiederholt bis genügend Kopien für die Sequenzierung vorliegen. Abschließend findet eine letzte Dentaurisierung des Clusters statt, da für die Sequenzierung Einzelstränge vorliegen müssen. [4]

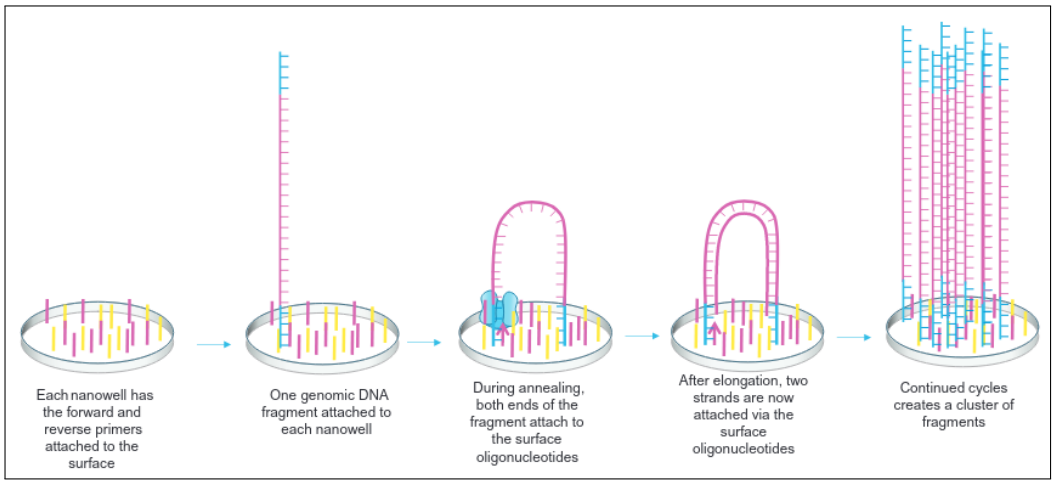

Abbildung 2.1: Bridge Amplification PCR [3]

2.1.3 Sequenzierung

Die Sequenzierung unter Illumina wird durch Sequenzierung mittels Synthese gleichzeitig für das alle Cluster auf der Fließzelle durchgeführt. Ein Primer wird an jedem Strang angelegt und eine Polymerase fügt fluoreszierend markierte, komplementäre Nukleotide (dNTP) zum Fragment hinzu [3]. Das hinzugefügte dNTP blockiert die weitere Vervollständigung des Komplementärstrangs und die Identität der Base kann über ein Bild der Färbung aufgezeichnet werden. Die Identifizierung einer Base wird auch Base Call genannt. [5]

Daraufhin wird die Blockierung chemisch entfernt und das nächste dNTP kann hinzugefügt und registriert werden [4].

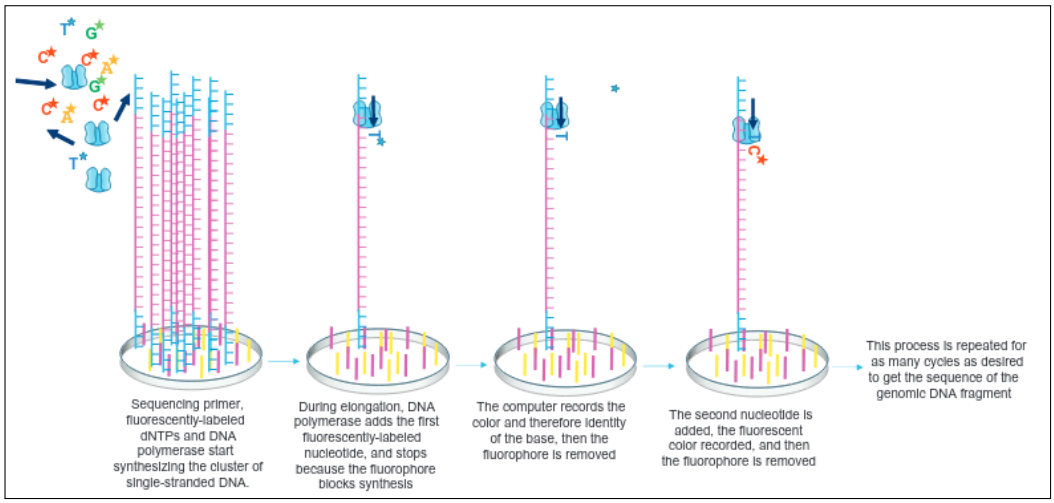

Abbildung 2.2: Sequencing by Synthesis [3]

Dieser Vorgang (vgl. Abbildung 2.2) wird wiederholt bis das Cluster vollständig abgelesen und der DNA-Abschnitt (engl.: Read) aufgezeichnet ist. Da eine einzige Fließzelle viele Millionen Cluster beinhalten kann, entstehen viele Millionen Reads aus einem Sequenzierungsvorgang. [3]

2.1.4 Datenanalyse

Das Illumina Sequenzierungsgerät verarbeitet die aufgenommenen Bilder der Sequenzierungszyklen direkt zu Base Calls und gibt diese als FASTQ-Datei aus. Diese enthält zusätzlich zur identifizierten Base einen zugehörigen Qualitätswert, der sich aus den Intensitäten der aufgenommenen Farben und deren Überlappungen berechnet. Die Qualität wird in einem Phred-Wert angegeben, welcher die logarithmische Fehlerwahrscheinlichkeit darstellt. [5]

In einem weiteren Schritt müssen die gewonnenen DNA-Daten zu einer kontinuierlichen Sequenz rekonstruiert werden. Dies kann entweder durch Abgleich mit einem Referenzgenom angepasst werden oder durch Vergleich der Reads miteinander erzielt werden, um Überlappungen festzustellen, üblicherweise mittels de novo assembly. [3]

2.2 FASTQ-Dateiformat

Die Grundlage für die Auswertung der Read-Qualitäten in der vorliegenden Arbeit sind die aus der Sequenzierung entstandenen FASTQ-Dateien. Diese beinhalten die Sequenzen und den zu jeder abgelesenen Base gehörigen Phred-Wert.

2.2.1 Aufbau

Eine FASTQ-Datei beinhaltet die Informationen zu allen Reads, die von den Clustern abgelesen wurden. Sie können mehrere Millionen Sequenzen umfassen und eine Größe von mehreren Megabyte bis Gigabyte haben. Jeder einzelne Read wird in vier Zeilen beschrieben.

Die erste Zeile dient dazu den Read zu identifizieren. Dabei beginnt sie stets mit einem '@'-Zeichen, gefolgt von einem Identifikator für die Sequenz. Für Sequenzen der Illumina Software ist eine Spezifikation festgelegt. Diese enthält Informationen zu dem verwendeten Instrument

und dessen Durchgangsnummer, der Fließzelle und den Nummern für Kanal (engl.: lane) und Platte (engl.: tile) dieser Sequenz, den x- und y-Koordinaten des Clusters, einer optionalen UMI-Sequenz[2], der Read-Nummer, ob eine Filterung stattgefunden hat, einer Kontrollnummer und dem Read-Index [7]. Diese Attribute werden in folgendem Format eingetragen:

```
@<Instrument>:<Durchgangsnummer>:<Fließzelle>:<Kanal>:<Platte>:
<x-Position>:<y-Position>:<UMI> <Read>:<Filterung>:<Kontrollnummer>:<Index>
```

Die zweite Zeile enthält die Buchstaben der Base Calls der Sequenz, also eine Zeichenfolge bestehend aus A, C, G, T und außerdem ein N für Basen, die nicht bestimmt werden konnten. Die dritte Zeile enthält nur ein „+"-Zeichen, um die Sequenz von ihren Qualitäten eindeutig zu trennen. In der vierten Zeile sind die Qualitäten, also Phred-Werte der einzelnen Base Calls, entsprechend der Reihenfolge der Sequenz aus Zeile 2, eingetragen. Die Phred-Werte sind dabei als ASCII-Zeichen kodiert. [8]

Ein vollständiger Eintrag für einen Read sieht damit exemplarisch folgendermaßen aus [7]:

```
1 @SIM:1:FCX:1:15:6329:1045:GATTACT+GTCTTAAC 1:N:0:ATCCGA
2 TCGCACTCAACGCCCTGCATATGACAAGACAGAATC
3 +
4 <>;##=><9=AAAAAAAAAA9#:<#<;<<<????#=
```

2.2.2 Phred-Wert

Die Qualität eines Base Calls wird im Phred-Wert Q_{Phred} erfasst. Dieser wird aus dem Logarithmus der Fehlerwahrscheinlichkeit P_e mit folgender Formel berechnet [8]:

$$Q_{Phred} = -10 \cdot \log_{10}(P_e) \tag{2.1}$$

Damit ergeben sich zum Beispiel aus der Fehlerwahrscheinlichkeit von 1 in 10 (bzw. einer Base Call Genauigkeit von 90%) ein Phred-Wert von 10 und für eine Fehlerwahrscheinlichkeit von 1 in 1000 (bzw. einer Base Call Genauigkeit von 99.9%) ein Phred-Wert von 30.

Für die Berechnung des Phred-Werts Q_{Phred} aus der Fehlerwahrscheinlichkeit P_e kann die Formel umgestellt werden:

$$P_e = 10^{\frac{Q_{Phred}}{-10}} \tag{2.2}$$

Vor der ASCII-Kodierung der Phred-Werte wird in der Regel noch ein Versatz (engl.: offset) hinzugefügt, da die ersten 32 ASCII-Zeichen Leerzeichen und nicht-darstellbare Symbole enthalten. Bei aktuellen Illumina-Verfahren beträgt dieser 33, bei älteren Datensätzen kann auch noch ein Versatz von 64 vorliegen. Dieser wird zum Phred-Wert hinzuaddiert und im ASCII-Zeichensatz kodiert. [8] Die Tabelle 2.1 zeigt die kodierten Zeichen und deren Fehlerwahrscheinlichkeit.

[2]Unique Molecular Identifier sind kurze Sequenzen, die während der Bibliotheksvorbereitung Molekülen hinzugefügt werden können, um die korrekte Anzahl an PCR-Duplikaten festzustellen[6].

Q_{Phred}	P_e	ASCII Phred+33	ASCII Phred+64	Q_{Phred}	P_e	ASCII Phred+33	ASCII Phred+64
0	1.00000	!	@	21	0.00794	6	U
1	0.79433	"	A	22	0.00631	7	V
2	0.63096	#	B	23	0.00501	8	W
3	0.50119	$	C	24	0.00398	9	X
4	0.39811	%	D	25	0.00316	:	Y
5	0.31623	&	E	26	0.00251	;	Z
6	0.25119	’	F	27	0.00200	<	[
7	0.19953	(	G	28	0.00158	=	\
8	0.15849	)	H	29	0.00126	>	]
9	0.12589	*	I	30	0.00100	?	^
10	0.10000	+	J	31	0.00079	@	_
11	0.07943	,	K	32	0.00063	A	‘
12	0.06310	-	L	33	0.00050	B	a
13	0.05012	.	M	34	0.00040	C	b
14	0.03981	/	N	35	0.00032	D	c
15	0.03162	0	O	36	0.00025	E	d
16	0.02512	1	P	37	0.00020	F	e
17	0.01995	2	Q	38	0.00016	G	f
18	0.01585	3	R	39	0.00013	H	g
19	0.01259	4	S	40	0.00010	I	h
20	0.01000	5	T	41	0.00008	J	i

Tabelle 2.1: ASCII Kodierung der Phred-Werte

2.3 Qualitätskontrolle mit FastQC

FastQC ist ein Programm zur Qualitätskontrolle der Sequenz-Datensätze aus FASTQ-Dateien (oder auch aus BAM- oder SAM-Dateien). Bevor Sequenzierungsdaten weiterverarbeitet und analysiert werden, kann FastQC durch Aggregation und Darstellung der Daten in verschiedenen Modulen einen Überblick über die Gesamtqualität der gewonnenen Reads verschaffen. Dadurch können Probleme im Sequenzierungsprozess oder Ausgangsmaterial festgestellt und die Tauglichkeit der Daten für eine Weiterverarbeitung sichergestellt werden. FastQC präsentiert die Ergebnisse in einem grafischen Bericht und in einem Textdokument und eignet sich damit sowohl zur direkten Analyse als auch zur Implementierung in einer bioinformatischen Pipeline. Der FastQC Bericht umfasst 11 Module mit einer automatisierten Bewertung jedes Moduls mit „normal“, „etwas abnormal“ oder „sehr ungewöhnlich“. Dies wird durch einen grünen Haken (im Textdokument: „pass“), orangenes Dreieck („warn“) oder rotes Kreuz („fail“) dargestellt. Die Module heißen „Basic Statistics“, „Per Base Sequence Quality“, „Per Sequence Quality Scores“, „Per Base Sequence Content“, „Per Sequence GC Content“, „Per Base N Content“, „Sequence Length Distribution“, „Duplicate Sequences“, „Overrepresented Sequences“, „Adapter Content“ und „Per Tile Sequence Quality“ [1]. Im vorliegenden Projekt wird untersucht, ob die automatisch generierten Bewertungen dieser Module zur Erstellung eines ML-Modells verwendet werden können. Das Modul „Basic Statistics“ wird ignoriert, da es keinem Bewertungsverfahren unterliegt.

3 Machine Learning Algorithmen

3.1 Machine Learning

Das Ziel von ML ist es Problemstellungen zu lösen, ohne dafür explizite Regeln programmieren zu müssen. Stattdessen erlernt ein ML-Algorithmus diese Regeln eigenständig, indem er auf Grundlage von Beispiel-Daten ein Modell aufbaut, welches ihm ermöglicht die richtige Lösung für neue Daten zu finden. Diese Beispiel-Daten werden als Trainingsdaten bezeichnet.

ML-Anwendungen sind vorteilhaft, wenn die Erstellung eines regelbasierten Programms unpraktikabel, teuer oder langsam ist. Dies kann zum Beispiel der Fall sein, wenn die Menge an Daten zu groß oder das Problem zu komplex ist, um eindeutige Regeln manuell feststellen zu können. Darüber hinaus können ML-Modelle sich flexibler an volatile Daten und verändernde Umgebungen anpassen. [9]

ML kann in zwei generelle Klassen eingeteilt werden - überwachtes und unüberwachtes Lernen. Überwachtes Lernen verwendet Trainingsdaten, deren Lösungen bekannt sind und zu jedem Datenpunkt als sogenannte Label vorliegen. Für dieses Projekt liegen gelabelte Daten vor und es wird überwachtes Lernen angewendet.

Eine typische Aufgabenstellung für ML ist die Klassifizierung. Dabei muss für jeden Datenpunkt eine Zuordnung zu einer definierten Klasse gefunden werden. Im vorliegenden Projekt existieren zwei Klassen - „good“ und „ugly“, welche die Gesamtqualität einer FASTQ-Datei bewerten. Dies sind damit die möglichen Label in diesem Projekt. Die Label wurden durch die manuelle Analyse von FastQC-Ergebnissen von Mitarbeitern des Robert Koch-Instituts erstellt und bereitgestellt. Die einzelnen Modulbewertungen der FastQC Software für eine FASTQ-Datei sind der Datenpunkt, genannt Features, des ML-Algorithmus.

Für Klassifizierungsprobleme existieren zahlreiche ML-Algorithmen. In dieser Arbeit werden die Algorithmen „Support Vector Machine“, „Decision Tree“ (deutsch: Entscheidungsbaum) und „Random Forest“ auf ihre Tauglichkeit hin untersucht, das gegebene Problem zu lösen. Diese Algorithmen werden im Folgenden vorgestellt.

3.2 Support Vector Machine

Eine Support Vector Machine (SVM) ist ein versatiler ML-Algorithmus, welcher für lineare und nichtlineare Klassifizierungsprobleme, aber auch für Regressionsprobleme und Außenseitererkennung eingesetzt werden kann. Sie eignen sich besonders gut für die Klassifizierung von kleinen aber komplexen Datensätzen. [9]

3.2.1 Lineare SVM Klassifizierung

Ein Datensatz ist linear separierbar, wenn die Datenpunkte als Vektoren in einem Vektorraum durch eine sogenannte Hyperebene voneinander getrennt werden können. Die Hyperebene ist ein Unterraum, welcher eine um eins kleinere Dimension hat als die Datenpunkte. Im Fall von

drei Dimensionen der Daten (also bei drei Features) ist die Hyperebene eine Ebene, welche die Datenpunkte in die Zielklassen teilt. Im Fall von zwei Dimensionen ist sie eine Gerade.

Das Ziel einer SVM ist es ebendiese Hyperebene zwischen den Klassen der Trainingsdaten zu ermitteln, welche den maximalen Abstand von den Klassen hat. Dafür muss der Abstand der Trainingsvektoren, die am nächsten zur Hyperebene liegen maximiert werden. Diese Vektoren werden als Stützvektoren (engl.: support vector) bezeichnet, ihr Abstand zur Hyperebene heißt Margin. Diese Begriffe sind in Abbildung 3.1 für den zweidimensionalen Fall dargestellt. Damit ist die Berechnung der SVM ein Optimierungsproblem. Der Nutzen einer maximalen Margin liegt darin auch Datenpunkte, die von den Trainingsdaten abweichen, richtig klassifizieren zu können. [9]

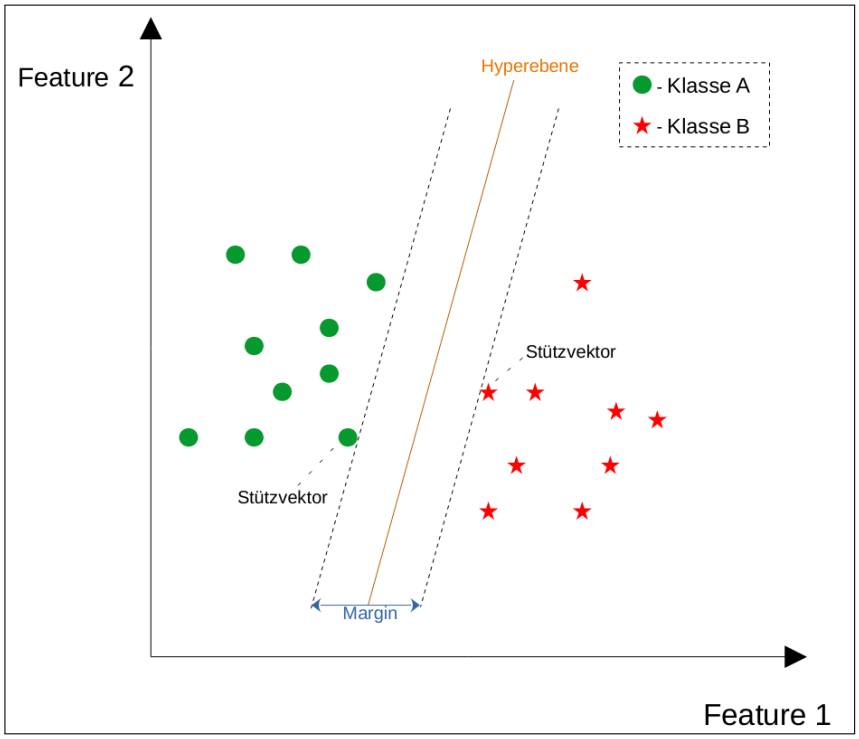

Abbildung 3.1: SVM im zweidimensionalen Raum (erstellt mit [10])

Der in Abbildung 3.1 dargestellte Fall, in dem alle Datenpunkte einer Klasse sich stets auf der jeweiligen Seite der Margin befinden müssen, nennt sich Hard Margin. Diese ist nur dann effektiv einsetzbar, wenn die Daten tatsächlich linear separierbar sind und keine groben Ausreißer enthält. Sollte sich also zum Beispiel ein Datenpunkt der Klasse A in der Region der Klasse B befinden, kann keine Hard Margin für den Datensatz gefunden werden. Dieses Problem kann mit einer Soft Margin gelöst werden. Hierbei gilt es, ein günstiges Gleichgewicht zwischen maximaler Breite der Margin und minimaler Anzahl der Margin-Verstöße[1] zu finden. [11]

[1]Unter Margin-Verstößen versteht man diejenigen Datenpunkte, die sich auf der falschen Seite der jeweiligen Margin befinden. Diese befinden sich entweder innerhalb der Margin oder im Bereich einer falschen Klasse.

3.2.2 Kernel-Funktion

Das Verfahren mit Soft Margin kann nie eine optimale Trennung der Daten finden, sondern nur einen Kompromiss zwischen den Fehlertypen ermöglichen. Durch die Erweiterung der SVM mit einer Kernel-Funktion lassen sich nichtlineare Klassifikationsprobleme in der Regel besser lösen. Das Vorgehen ist dabei die Daten in einem Raum höherer Dimension abzubilden, indem man ein zusätzliches Feature auf Grundlage der vorhandenen Features durch eine Kernel-Funktion konstruiert. In dem Raum dieser höheren Dimension kann dann eine Hyperebene gefunden werden, welche die Klassen voneinander trennt. Der Erfolg hängt dabei vor allem von der Wahl der passenden Funktion ab. [11]

Abbildung 3.2 zeigt ein Beispiel für einen Datensatz, der im zweidimensionalen Raum nicht linear separierbar ist, im dreidimensionalen Raum jedoch vollständig die Klassen voneinander trennen kann.

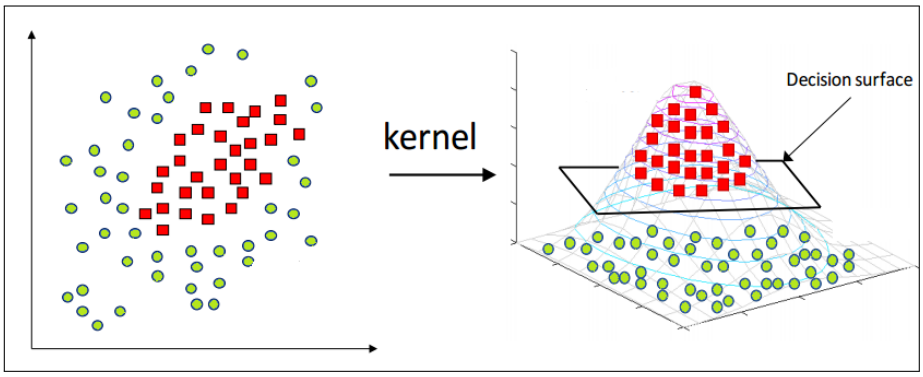

Abbildung 3.2: Anwendung einer Kernel-Funktion zur Auffindung der Hyperebene [12]

3.3 Decision Tree

Ein Decision Tree ist ein gerichteter Entscheidungsbaum, der unter anderem zur Klassifizierung eingesetzt werden kann. Jeder Knoten des Baums ist ein Feature-Test und die aus dem Knoten entspringenden Kanten die möglichen Ergebnisse. Die Blätter des Baums enthalten die Klassifikation, die aus den vorangegangen Knoten und Kanten resultiert. Um die Klassifikation eines Datenpunkts zu erhalten, wird der Decision Tree durchlaufen bis man bei einem Blatt ankommt und damit das Ergebnis erhält. Diese Eigenschaften sind in Abbildung 3.3 beispielhaft dargestellt. Die Feature-Tests zerlegen den Raum der Datenpunkte so, dass jeder Datenpunkt genau einer Klasse zugeordnet werden kann. Um einen Decision Tree für einen Datensatz zu erlernen durchsucht der Algorithmus alle möglichen Feature-Tests und wählt denjenigen, welcher den größten Informationsgehalt bezüglich der Klassen aufweist, also die Datenpunkte einer Klasse von den Datenpunkten der anderen am besten trennt. Besitzen die Datenpunkte in einem Ergebnisset alle die gleiche Klasse, wird dies zu einem reinen Blatt des Baums. Wenn eine Ergebnismenge noch nicht uniform ist, kann das Verfahren für die darin enthaltenen Datenpunkte rekursiv fortgeführt werden. Der Algorithmus kann solange durchgeführt werden, bis nur noch reine Blätter resultieren. In dem Fall hätte der Decision Tree für den Trainingsdatensatz eine Genauigkeit von 100%. In der Regel impliziert dies ein zu komplexes Modell und eine Überanpassung (engl.: Overfitting) der Trainingsdaten, was mit einer schlechteren Anpassungsgüte (engl.: Performance) des ML-Modells für neue Datenpunkte einhergeht. [13]

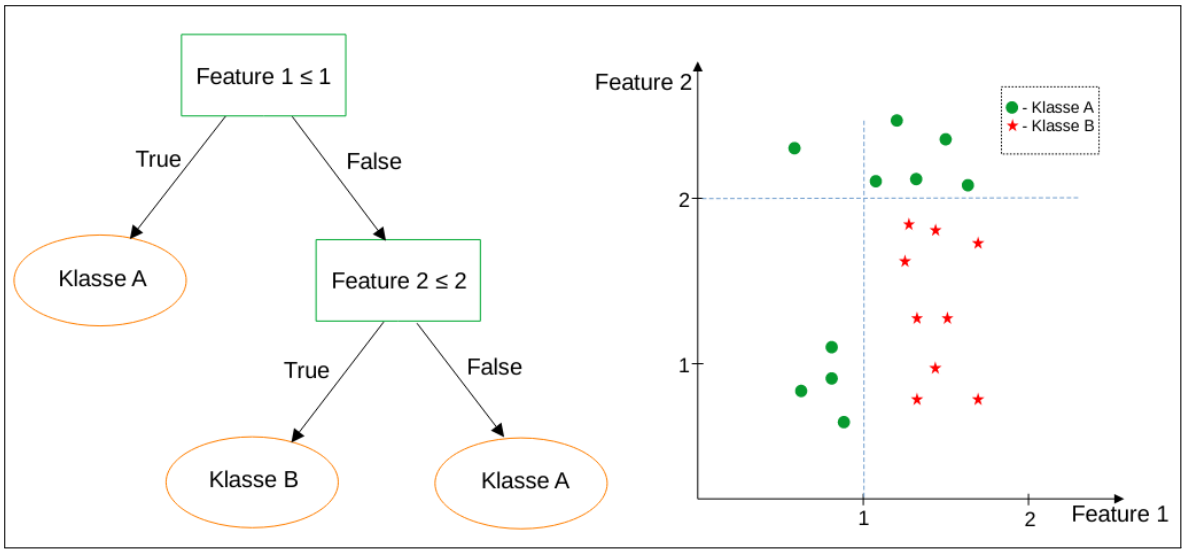

Abbildung 3.3: Entscheidungsbaum und dessen Raumzerteilung (erstellt mit [10])

3.3.1 Pruningverfahren

Dem Overfitting kann man mit Pruningverfahren entgegenwirken, dem Pre-Pruning oder dem Post-Pruning. Für das Pre-Pruning können vor der Baumerstellung Kriterien festgelegt werden, die den Algorithmus vorzeitig stoppen. So kann man ihn durch eine Festlegung einer maximalen Tiefe des Baums oder Anzahl an Blättern anhalten oder eine Mindestanzahl von Datenpunkten für die Erstellung eines neuen Knotens voraussetzen. Das Post-Pruning kann durchgeführt werden, indem man nach der Fertigstellung des Baums Knoten entfernt, die wenig Information enthalten. [13]

3.3.2 Gini-Index

Als Maß für den Informationsgehalt einer Teilung wird üblicherweise der Gini-Index verwendet. Dieser gibt die Wahrscheinlichkeit wieder, bei zufälliger Auswahl, einen falsch zugeordneten Datenpunkt zu ziehen. Ein reines Blatt hat einen Gini-Index von 0, ein Knoten mit einer gleichen Verteilung der Klassen 0,5. Sei D ein Datensatz mit Datenpunkten in K Klassen und die Wahrscheinlichkeit eines Datenpunkts an einem Knoten zur Klasse i zu gehören p_i, so kann der Gini-Index G von D definiert werden als [14]:

$$G(D) = 1 - \sum_{i=1}^{K} p_i^2 \tag{3.1}$$

Um den idealen Feature-Test für einen Knoten zu bestimmen, muss der Test mit dem kleinsten gewichteten Gini-Index gewählt werden. Wenn ein Datensatz D von Größe n an einem Feature-Test A in zwei Mengen D_1 und D_2 mit jeweils n_1 und n_2 Anzahl an Datenpunkten geteilt wird, kann der gewichtete Gini-Index $G_A(D)$ berechnet werden mit [14]:

$$G_A(D) = \frac{n_1}{n} G(D_1) + \frac{n_2}{n} G(D_2) \tag{3.2}$$

3.4 Random Forest

Ein grundlegendes Problem von Decision Trees ist das Overfitting der Trainingsdaten. Sie passen ihre Entscheidungskriterien also zu genau an die Daten an, durch die sie aufgebaut wurden und können neue Datenpunkte häufig nicht mit einer vergleichbaren Genauigkeit klassifizieren. Auch die Pruningverfahren helfen hier nur bedingt und verbessern die Generalisierbarkeit des Modells nur wenig. Ein weitaus effektiveres Verfahren zur Lösung dieses Problems ermöglicht der ML-Algorithmus Random Forest.

Ein Random Forest wird aus einer Mehrzahl von Decision Trees aufgebaut. Dabei wird für die Erstellung jedes Decision Trees nur ein zufälliger Teil der Daten und ein zufälliger Teil aller Features verwendet. Die Idee hierbei ist, dass jeder einzelne Decision Tree seine Trainingsdaten overfitten kann, aber der Konsens aus ihren Einzelergebnisse eine weitaus bessere Generalisierbarkeit bietet als ein einzelner Decision Tree. [13]

Die zufällige Wahl der Daten für jeden Decision Tree ist ein sogenanntes Bootstrap Sample. Dafür werden aus n Daten n mal zufällig Datenpunkte mit Zurücklegen gewählt. Damit ist der Datensatz genauso groß wie der ursprüngliche, enthält aber nicht alle Datenpunkte, dafür aber andere wiederholt. Für jede Wahl eines Feature-Tests für einen Knoten wird nur eine zufällig gewählte Teilmenge aller Features betrachtet und der beste gewichtete Gini-Index unter diesen ermittelt. Diese Teilmenge ist für jeden Knoten eines Decision Trees unterschiedlich. Durch diese beiden Verfahren sind alle Decision Trees im Random Forest verschieden. [13]

Die Parameter für die Erstellung eines Random Forest sind somit die Anzahl der Bäume und die Anzahl der Features, die für jeden Knoten in Betracht gezogen werden sollen.

Für die Klassifizierung eines Datenpunktes im Random Forest werden die Klassifikationen aller Decision Trees für ihn aggregiert. Die Klassifikation des Random Forests ist dann diejenige, die am häufigsten von allen Decision Trees vorgeschlagen wurde. [9]

3.5 Datenvorbereitung für die ML-Modelle

Für die Implementierung der ML-Modelle liegen in diesem Projekt 184 Sequenzierungsdaten im FASTQ-Format vor. Diese bestehen aus 88 Sequenzierungen von Enterococcus faecium (E. faecium) DNA, 18 Sequenzierungen von Escherichia coli (E. coli) DNA und 78 Sequenzierungen von Staphylococcus aureus (S. aureus) DNA. Es werden sowohl ML-Modelle für den gesamten Datensatz, als auch einzelne Modelle für jede Bakterienart implementiert, um auch mögliche Unterschiede in den Qualitätseinschätzungen der Ausgangsorganismen zu berücksichtigen.

Die FASTQ Dateien werden zunächst durch die FastQC-Software analysiert, um dann die automatischen Bewertungen (Status) der FastQC-Module und die grundlegenden Daten aus den generierten „Basic Statistics“ zu extrahieren. Diese bilden die Features mit denen die ML-Modelle trainiert werden. Die Label sind die manuellen Gesamtbewertungen durch Biologen und fallen in die zwei Klassen „good“ und „ugly“. Um die Modulbezeichnungen in der Arbeit zu vereinfachen, wurden sie durchnummeriert. In der Tabelle 3.1 sind alle verwendeten Features und deren entsprechende Erklärung aufgelistet.

Damit Features in einem ML-Algorithmus verwendet werden können, müssen sie in einem zahlenmäßigen Format vorliegen. Dies ist für die Features der „Basic Statistics“-Attribute bereits gegeben. Die einzelnen Statusvarianten, welche einen Wert aus „pass“, „warn“ oder „fail“ annehmen, müssen zunächst codiert werden (engl.: feature encoding). Da die möglichen Werte eine

Feature	Erklärung
total_sequences	Anzahl der Sequenzen
percent_gc	Gesamter GC-Anteil
min_sequence_length	Kürzeste Sequenzlänge
max_sequence_length	Längste Sequenzlänge
module_1_status	Per Base Sequence Quality Status
module_2_status	Per Tile Sequence Quality Status
module_3_status	Per Sequence Quality Scores Status
module_4_status	Per Base Sequence Content Status
module_5_status	Per Sequence GC Content Status
module_6_status	Per Base N Content Status
module_7_status	Sequence Length Distribution Status
module_8_status	Sequence Duplication Levels Status
module_9_status	Overrepresented Sequences Status
module_10_status	Adapter Content Status

Tabelle 3.1: Verwendete Features

eindeutig geordnete Beziehung zueinander haben, bietet sich eine Ordinal-Codierung an. Dazu wird jedem Wert ein geordneter Integer zugeordnet und im Datensatz ersetzt: „fail“ mit 0, „warn“ mit 1 und „pass“ mit 2.

Damit ist die Vorbereitung der Daten abgeschlossen und sie können zum Training der ML-Modelle verwendet werden.

3.6 Evaluierung von Machine Learning Modellen

Grundlegend für die Auswahl des besten ML-Modells und dessen Parametern ist es, die Performance des Modells für neue Datenpunkte evaluieren zu können. Das bedeutet, dass von den gegebenen Daten nicht alle zum Training verwendet werden sollten, sondern nur ein Teil (Trainingsdaten). Der andere Teil wird zurückgehalten und kann dann verwendet werden um das fertige ML-Modell zu testen (Testdaten) und verschiedene Performance-Kennzahlen zu berechnen.

3.6.1 Kreuzvalidierung

Die Teilung der Daten kann einmalig durchgeführt werden oder wiederholt in der sogenannten Kreuzvalidierung. Der Nachteil der einmaligen Teilung besteht darin, Informationen aus den Testdaten für das Erlernen des ML-Modells zu verlieren. Das kann sich besonders bei kleinen Datensätzen auf die Qualität des Modells auswirken, wodurch die Evaluierung zu schlechteren Performance-Kennzahlen führt als bei einem Modell für die gesamten Daten tatsächlich vorliegen würden. Die Kreuzvalidierung kann dieses Problem lösen und zuverlässigere Performance-Kennzahlen berechnen.

Dafür wird der gesamte Datensatz bei einer k-fachen Kreuzvalidierung in k möglichst gleich große Teile zerlegt. Danach werden k Testdurchläufe ausgeführt, indem jeweils ein Teil als Testdatensatz verwendet wird und alle restlichen Teile die Trainingsdaten eines ML-Modell sind. Damit

werden also k verschiedene Modelle erstellt, die k verschiedene Performance-Kennzahlen produzieren. Für das Endergebnis einer Kennzahl kann ihr Durchschnitt und Varianz aller Durchläufe berechnet werden. [13]

Die Anzahl k kann so hoch wie notwendig festgelegt werden, der Nachteil liegt in einem höheren Rechenaufwand, da umso mehr Modelle trainiert werden müssen. Üblicherweise verwendet man ein k von 5-10 für einen guten Kompromiss zwischen Zuverlässigkeit der Kennzahlen und Dauer der Ausführung [13]. Der Extremfall ist die Leave-One-Out-Kreuzvalidierung, bei der k gleich der Anzahl N der Datenpunkte im gesamten Datensatz ist. Das heißt, jedes Modell der Kreuzvalidierung verwendet $N - 1$ Daten für sein Training und erstellt N Modelle. Die Ausführung ist besonders rechen- beziehungsweise zeitaufwendig. Dafür ist die Berechnung der Kennzahlen am genauesten, hat aber eine höhere Varianz.

3.6.2 Kennzahlen

Für die Bewertung eines ML-Modells zur binären Klassifizierung existieren verschiedene Kennzahlen. Das Grundprinzip zur Ermittlung der Kennzahlen liegt darin, die Datenpunkte der Testdaten durch das ML-Modell klassifizieren zu lassen und mit dem vorliegenden, tatsächlichen Label zu vergleichen um festzustellen, ob eine korrekte Klassifikation oder ein Fehler vorliegt. Die naheliegendste Performance-Kennzahl ist die Genauigkeit (engl.: Accuracy), sie beschreibt den Anteil der richtigen Klassifikationen, siehe Formel 3.3. Für viele Anwendungsfälle ist dieser Wert jedoch nicht sehr aussagekräftig, insbesondere bei einer Unausgewogenheit der Zielklassen. So würde beispielsweise ein Klassifikator, der jeden Datenpunkt immer nur einer Klasse zuweist, bei einem Datensatz der zu 95% nur Datenpunkte ebendieser Klasse enthält eine Accuracy von 95% besitzen, obwohl er nicht in der Lage ist die andere Klasse zu identifizieren. Um diese Qualität besser ausdrücken zu können, sollten nicht nur Fehler im Allgemeinen, sondern auch deren Typ berücksichtigt werden. So kann eine falsche binäre Klassifikation falsch negativ oder falsch positiv sein, je nachdem ob es das positive Ereignis beziehungsweise das negative Ereignis falsch klassifiziert hat. Die Konfusionsmatrix in Tabelle 3.2 zeigt diese Beziehung im Kontext der Klassen dieses Projekts.

	Tatsächlich „good“	Tatsächlich „ugly“
Klassifikation als „good“	richtig positiv t_p	falsch positiv f_p
Klassifikation als „ugly“	falsch negativ t_n	richtig negativ f_n

Tabelle 3.2: Konfusionsmatrix

Aus der Konfusionsmatrix lassen sich die folgenden Performance-Kennzahlen ableiten, die für unterschiedliche Anwendungen relevant sein können [13]:

$$Accuracy = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \tag{3.3}$$

$$Precision = \frac{t_p}{t_p + f_p} \tag{3.4}$$

$$Recall = \frac{t_p}{t_p + f_n} \tag{3.5}$$

$$FPR = \frac{f_p}{f_p + t_n} \tag{3.6}$$

Der Precision-Wert (siehe Formel 3.4) misst den Anteil der tatsächlich positiven Datenpunkte von allen als positiv klassifizierten Datenpunkten. Er ist relevant, wenn das Ziel der Klassifizierung ist, die Anzahl der falsch positiven Klassifikationen zu begrenzen. Der Recall (deutsch: Sensitivität, siehe Formel 3.5) beschreibt den Anteil der richtig positiv klassifizierten Datenpunkte von allen tatsächlich positiven Datenpunkten. Dieser ist für Anwendungen relevant, deren falsch negativen Ergebnisse minimiert werden sollen. [13]

Die falsch positiv-Rate (FPR, siehe Formel 3.6) gibt den Anteil der falsch positiven Klassifikationen von allen tatsächlich negativen Datenpunkten wieder. Häufig ist auch die Kombination aus Precision und Recall für den Anwendungsfall von Relevanz, insbesondere, wenn man verschiedene Klassifizierungen miteinander vergleichen will. Dazu kann das harmonische Mittel beider Werte verwendet werden, ausgedrückt im F_1-Score [9]:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{t_p}{t_p + \frac{f_n + f_p}{2}} \tag{3.7}$$

Durch ihre Definition sind Precision und FPR voneinander abhängig. Möchte man ein ML-Modell für einen der Werte optimieren, muss man Abzüge im anderen hinnehmen. Diese Beziehung kann durch die ROC-Kurve (receiver operating characteristic) dargestellt werden. Die blau eingezeichnete Kurve in Abbildung 3.4 ist die ROC-Kurve eines Klassifizierungsmodells und zeigt die Abhängigkeit der beiden Werte voneinander. Je näher die Kurve an der Ecke links oben liegt, desto besser ist die Gesamtperformance. Eine zufällige Klassifizierung ist durch die schwarz gestrichelte Gerade dargestellt. Der Gesamtwert der ROC-Kurve kann durch die Fläche unter ihr zusammengefasst werden, dem AUC-Wert (area under the curve). Bei einer perfekten Klassifizierung liegt dieser bei 1, bei einer zufälligen bei 0,5. Der AUC-Wert eignet sich gut, um verschiedene ML-Modelle miteinander zu vergleichen. [9]

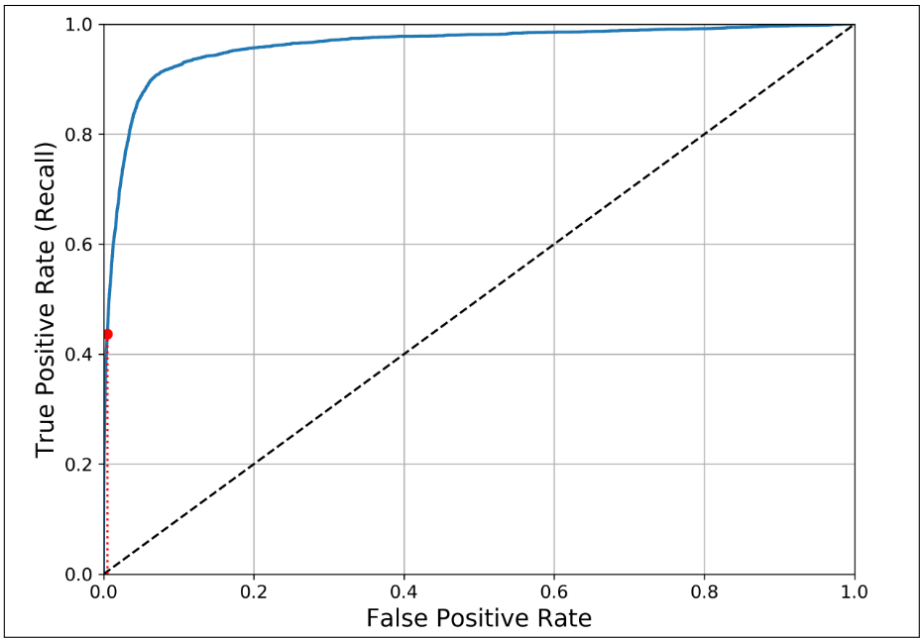

Abbildung 3.4: ROC-Kurve [9]

4 Ergebnisse

4.1 Ergebnispräsentation

Mit den gegebenen Sequenzierungsdaten wurden die oben erklärten ML-Modelle implementiert: SVM, Decision Tree und Random Forest. Diese wurden zum einen mit dem vollständigen Datensatz erstellt, zum anderen für jede einzelne Bakterienart. Die Ergebnisse werden im Folgenden für die Modelle mit dem vollständigen Datensatz detailliert dargestellt und abschließend tabellarisch mit den Ergebnissen bei Verwendung der jeweiligen Bakterienart verglichen.

4.1.1 Korrelationsmatrix

Bevor die Modelle trainiert werden, kann eine Korrelationsmatrix der Features dargestellt werden, um Aufschluss über die Beziehungen der einzelnen Features miteinander (und dem Label) zu geben, dargestellt in Abbildung 4.1. Eine positive Korrelation zweier Features, rot dargestellt, bedeutet, dass ein Anwachsen des einen Features mit dem Anwachsen des anderen Features einhergeht. Eine negative Korrelation, blau dargestellt, bedeutet, dass mit dem Anwachsen des einen Features eine Abnahme des anderen Features auftritt. Diese Informationen sind in erster Linie in Verbindung mit der Label-Variable (target) von Bedeutung, da sie eine erste Information darüber geben, welche Features für die Klassifizierung am relevantesten sind. Die Korrelation der Features untereinander kann außerdem dazu dienen, redundante Informationen zu identifizieren und in zukünftigen Betrachtungen wegzulassen.

Von besonderer Bedeutung für das Label scheinen demnach die Variablen *module_1_status* und *module_2_status* zu haben, wobei auch bei den Variablen *total_sequences*, *percent_gc*, *module_8_status* und *module_10_status* eine eindeutige Korellation mit dem Label vorliegt. Weiterhin auffällig ist die Korrelation zwischen den Features *min_sequence_length*, *module_7_status* und *module_10_status*, die für eine spätere Reduzierung der Features infrage kommen könnten.

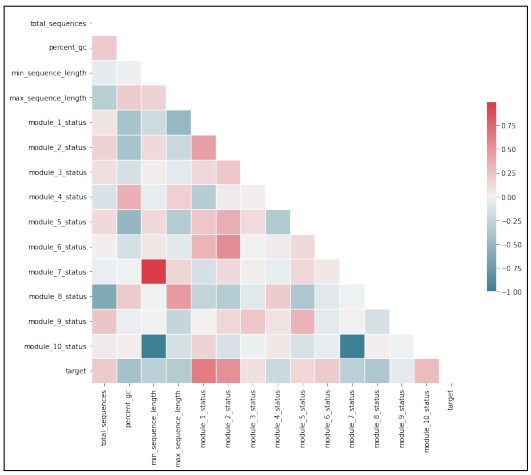

Abbildung 4.1: Heatmap der Feature-Korrelation (erstellt mit [15])

4.1.2 SVM Ergebnisse

Die SVM ist das rechenaufwändigste der drei ML-Modell, benötigt also die längste Zeit zum Training. Daher wurde die Kreuzvalidierung auf $k = 10$ beschränkt. Eine der entstehenden ROC-Kurven ist in Abbildung 4.2 dargestellt.

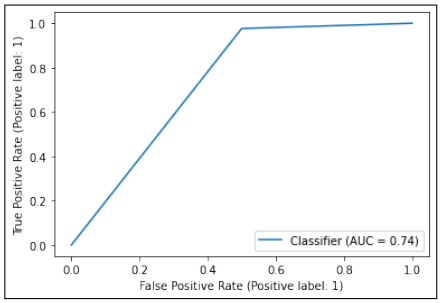

Abbildung 4.2: ROC-Kurve der SVM (erstellt mit [16])

Die Kennzahlen aus dem Ergebnis der Kreuzvalidierung für den gesamten Datensatz können der Tabelle 4.1 entnommen werden.

	Accuracy	Precision	Recall	F_1-Score	AUC
Durchschnitt	0.68	0.67	0.95	0.78	0.74
Standardabweichung	0.14	0.11	0.14	0.10	-

Tabelle 4.1: Ergebnisse der SVM Kreuzvalidierung (gesamter Datensatz)

Damit ist die Performance der SVM mit linearem Kernel akzeptabel. Es kann neue Datenpunkte wesentlich besser klassifizieren als eine zufällige Klassifizierung und erzielt einen sehr guten Recall-Wert, dafür sind Precision und Accuracy deutlich schlechter. Durch Implementierung einer komplexeren Kernel-Funktion könnte möglicherweise auch die Precision und die Gesamtperformance weiter verbessert werden. Dies wurde jedoch aufgrund des wesentlich recheintensiveren Trainings und der Ergebnisse der folgendenen beschriebenen Modelle nicht weiter untersucht.

4.1.3 Decision Tree Ergebnisse

Der Decision Tree ist der schnellste der drei Algorithmen und konnte hervorragende Ergebnisse für die Sequenzierungsdaten erreichen. Die ROC-Kurve für eins der in der Kreuzvalidierung erstellten Modelle ist in 4.3 abgebildet.

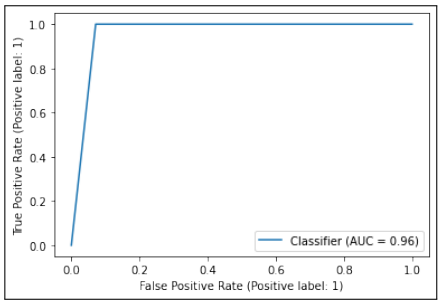

Abbildung 4.3: ROC-Kurve des Decision Trees (erstellt mit [16])

Mit einem AUC von 0.96 ist die Performance des Decision Trees damit der der SVM deutlich überlegen. Auch die anderen Kennzahlen zeigen, dass mit ihm eine außerordentlich gute Klassifizierung erzielt werden kann, siehe Tabelle 4.2.

	Accuracy	Precision	Recall	F_1-Score	AUC
Durchschnitt	0.93	0.93	0.98	0.95	0.96
Standardabweichung	0.08	0.11	0.05	0.06	-

Tabelle 4.2: Ergebnisse der Decision Tree Kreuzvalidierung (gesamter Datensatz)

Ein besonderer Vorteil des Decision Trees ist seine Eigenschaft, sein Klassifizierungsverhalten besonders verständlich darstellen zu können. Die Visualisierung 4.4 zeigt den vollständigen Decision Tree mit den gewählten Feature-Tests für jeden Knoten und dem für den Test ermittelten gewichteten Gini-Index. Mit dem Wert *samples* wird für jeden Knoten und Blatt die Anzahl der Datenpunkte angegeben, die sich aus dem vorangegangenen Feature-Test ergeben haben. Die Angaben unter *value* listen auf, wie viele Datenpunkte sich von jeder Klasse an dem Knoten oder Blatt befinden. An erster Position die Anzahl der Datenpunkte mit dem Label „ugly“, an zweiter die Anzahl mit dem Label „good“. Die Klasse, die an dem jeweiligen Knoten oder Blatt überwiegt, ist sowohl farblich mit orange („ugly“) oder blau („good“) gekennzeichnet, als auch als *class* beschrieben.

Die linke Kante eines Knotens ist stets das positive und die rechte Kante das negative Ergebnis des Feature-Tests.

Daraus folgt für das Beispiel des Wurzelknotens: es wird der Test für das Feature *module_1_status* ausgewählt mit der Frage, ob dieser Wert kleiner gleich 0,5 ist. Dieser Test wird gewählt, weil er von allen möglichen Tests den niedrigsten Gini-Index aufweist und die Daten am besten trennt. An diesem Knoten wird also gefragt, ob das Modul „Per Base Sequence Quality“ von FastQC mit „fail“ bewertet wurde. Wenn dies der Fall ist, folgt man der linken Kante und landet bei einem reinen Blatt, in dem alle 32 Datenpunkte das Label „ugly“ besitzen. Wenn der Test negativ beantwortet wird, folgt man der rechten Kante und landet bei einem neuen Knoten, der 28 Datenpunkte mit dem Label „ugly“ und 68 Datenpunkte mit dem Label „good“ enthält. Für diese wird der neue beste Feature-Test ausgeführt, welcher testet, ob die Anzahl der Sequenzen in einem Datenpunkt kleiner gleich 578143 ist.

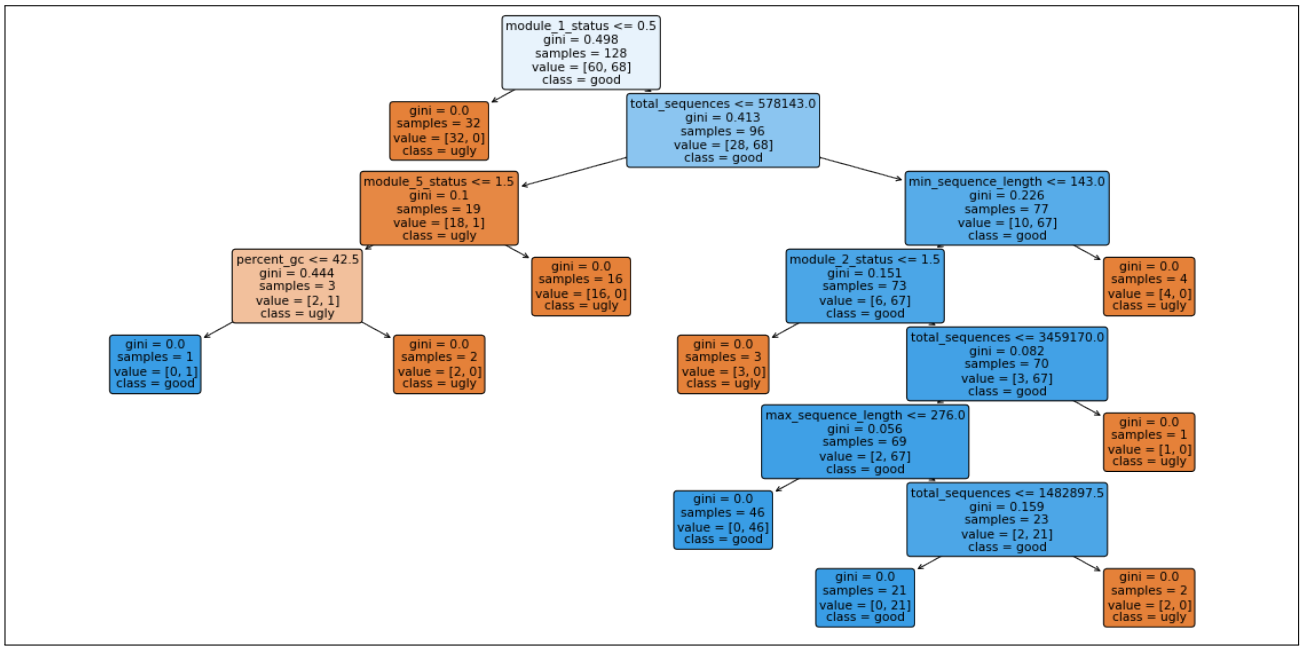

Abbildung 4.4: Visualisierung des Entscheidungsbaums (erstellt mit [16])

Nach einer Tiefe von 7 Kanten sind alle Datenpunkte des Testdatensatzes eindeutig eingeordnet und es liegen nur noch reine Blätter vor. Das bemerkenswerte an dem erstellten Entscheidungsbaum ist, dass er sich auch sehr gut auf ungesehene Daten übertragen lässt, wie die Kreuzvalidierung zeigt. Dies lässt sich damit erklären, dass alle im Projekt verfügbaren Daten sehr klar voneinander zu trennen sind, also keine besonders schwierig zu evaluierenden Sequenzierungsdaten enthalten sind.

4.1.4 Random Forest Ergebnisse

Es sollte weiterhin untersucht werden, ob ein Random Forest eine noch bessere Performance erzielen kann. Dies könnte besonders in zukünftigen Untersuchungen mit neuen Datensätzen von Bedeutung sein, wenn diese sich von einem Decision Tree nicht so zuverlässig klassifizieren lassen. Dieser wird mit einer Anzahl von 100 Bäumen erstellt. Die ROC-Kurve eines der Random Forest-Modelle der Kreuzvalidierung ist in Abbildung 4.5 dargestellt. Wie zu erwarten war, ist die Performance noch besser als bei einem Decision Tree mit einer AUC von 1.0, also einer perfekten Klassifizierung.

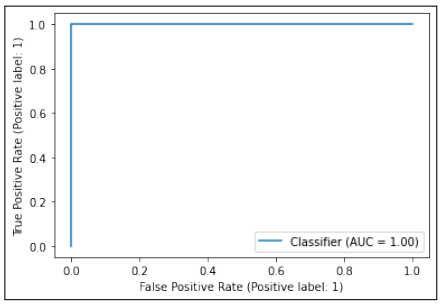

Abbildung 4.5: ROC-Kurve des Random Forests (erstellt mit [16])

Auch die Werte der Accuracy, Precision und F_1-Scores weisen auf eine weitere Verbesserung der Performance hin, dargestellt in Tabelle 4.3.

	Accuracy	Precision	Recall	F_1-Score	AUC
Durchschnitt	0.96	0.96	0.98	0.97	1.0
Standardabweichung	0.06	0.07	0.05	0.05	-

Tabelle 4.3: Ergebnisse der Random Forest Kreuzvalidierung (gesamter Datensatz)

Die Darstellung eines der 100 Bäume des Random Forests bietet nur wenig praktischen Nutzen, sie laufen schematisch alle ähnlich wie der Entscheidungsbaum ab, verwenden jedoch nur jeweils ein Bootstrap Sample der Daten und eine Teilmenge der Features.

Eine aufschlussreichere Darstellung der Erkenntnisse des Random Forest-Modells ist die Feature Importance. In Abbildung 4.6 ist dargestellt, welche der Features in allen 100 Bäumen den größten Einfluss darauf hatten, die beiden Zielklassen voneinander zu trennen. Die Features *total_sequences* und *module_1_status* sind mit Abstand am entscheidensten, wobei *module_2_status* und *percent_gc* auch noch einen großen Einfluss ausüben. Die Status für die Module 3, 6 und 9 haben kaum eine Bedeutung. Man kann in zukünftigen Weiterentwicklungen evaluieren diese aus dem Modell zu entfernen.

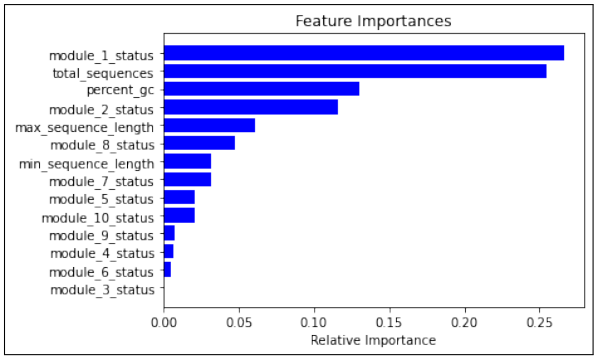

Abbildung 4.6: Feature Importance des Random Forests (erstellt mit [17])

4.1.5 ML-Modelle je Organismus

Im vorliegenden Datensatz befinden sich Sequenzierungsdaten zu drei unterschiedlichen Bakterienarten. Es soll untersucht werden, ob es sinnvoll sein kann für jede ein eigenes ML-Modell zu trainieren.

Die Implementation der Algorithmen für den gesamten Datensatz kann unverändert übernommen werden mit der Einschränkung, nur die Daten einer Bakterienart zu verwenden. Dabei ist von vornherein ein entscheidender Nachteil anzumerken - zum Training sind damit sehr viel weniger Daten verfügbar. Besonders im Fall der E. coli DNA liegen nur 18 Sequenzierungsdaten vor, wovon nur 2 als „good“ gelabelt sind, Dies kann ein Training unmöglich machen, wenn im zufälligen Trainingsdatensatz keine „good“ gelabelten Daten vorhanden sind. Gleichzeit sind die Ergebnisse der Tests nicht aussagekräftig, wenn sie nur über eine Klasse verfügen. Darausfolgend können die Ergebnisse des E. coli-Modells ignoriert werden und sind nur zur Vollständigkeit der Daten mit aufgeführt und mit einem „*“ versehen. Die Ergebnisse aller Modelle sind in der folgenden Tabelle 4.4 dargestellt.

Modell	Accuracy	Precision	Recall	F_1-Score	AUC
Gesamt (SVM)	0.68	0.67	0.95	0.78	0.74
E. faecium (SVM)	0.86	0.86	0.98	0.91	0.6
S. aureus (SVM)	0.62	0.73	0.62	0.63	0.69
E. coli (SVM)*	1.0*	0.4*	0.4*	0.4*	-
Gesamt (Decision Tree)	0.93	0.93	0.98	0.95	0.96
E. faecium (Decision Tree)	0.98	1.0	0.96	0.97	1.0
S. aureus (Decision Tree)	0.96	0.97	0.98	0.97	1.0
E. coli (Decision Tree)*	1.0*	0.2*	0.2*	0.2*	-
Gesamt (Random Forest)	0.96	0.96	0.98	0.97	1.0
E. faecium (Random Forest)	1.0	1.0	1.0	1.0	1.0
S. aureus (Random Forest)	0.99	1.0	0.98	0.99	1.0
E. coli (Random Forest)*	1.0*	0.2*	0.2*	0.2*	-

Tabelle 4.4: Vergleich der Ergebnisse aller ML-Modelle

Insgesamt lässt sich eine Tendenz erkennen, dass Organismus spezifische Modelle eine etwas bessere Performance erzielen, insbesondere beim Decision Tree und Random Forest. Desweiteren sind Unterschiede zwischen den Modellen in der Feature Importance der verschiedenen Random Forests festzustellen. Während *total_sequences* bei allen Modellen stets einen der beiden ersten Plätze belegt, ist *module_1_status* für die Klassifizierung im S. aureus-Modell von geringerer Bedeutung. Dafür ist beim S. aureus der *module_2_status* derjenige mit der höchsten Feature Importance. Das *percent_gc* Feature ist für den Random Forest der Gesamtdaten von großer Bedeutung. Es ist bei allen Organismus spezifischen Modellen jedoch weitaus weniger relevant. Die Unterschiede im Detail können den Abbildungen 4.6, 4.7, 4.8 und 4.9 entnommen werden.

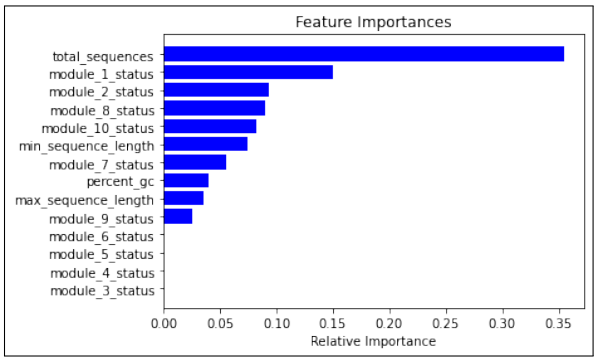

Abbildung 4.7: Feature Importance des E. faecium Random Forests (erstellt mit [17])

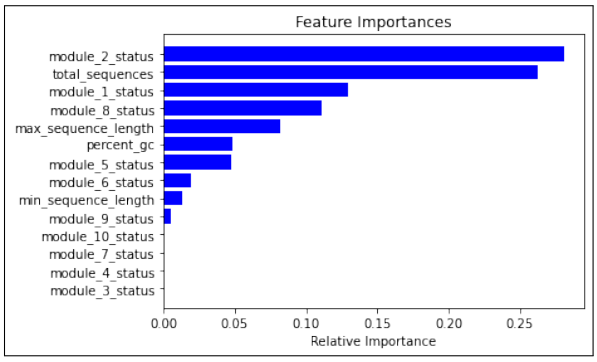

Abbildung 4.8: Feature Importance des S. aureus Random Forests (erstellt mit [17])

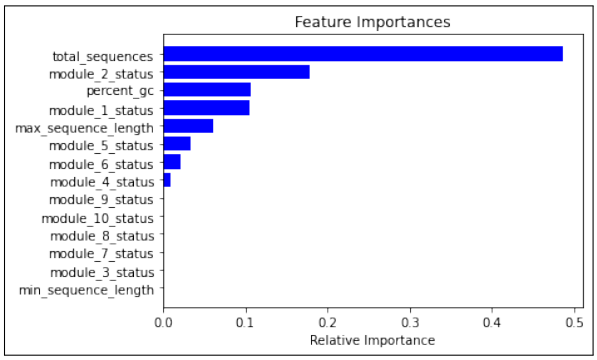

Abbildung 4.9: Feature Importance des E. coli Random Forests (erstellt mit [17])

4.2 Zusammenfassung

In der Umsetzung dieses Projekts konnte dargelegt werden, dass die Evaluierung der Qualität von Sequenzierungsdaten durch ML-Algorithmen umgesetzt werden kann. Die ML-Modelle konnten erfolgreich mit einfachen Features, die aus der FastQC Analyse der FASTQ-Dateien extrahiert wurden, trainiert werden und erzielten für die vorliegenden Daten akzeptable bis hervorragende Ergebnisse in einer Kreuzvalidierung. Die allerbesten Resultate konnten erzielt werden, indem Random Forests für spezifische Organismen implementiert wurden, was darauf schließen lässt, dass die Qualitätsmerkmale in den Sequenzierungsdaten unterschiedlicher Organismen von unterschiedlicher Bedeutung sind. Doch auch die Modelle des Decision Trees und Random Forests für den gesamten Datensatz, ohne Beachtung der Art des Organismus, führt zu nahezu perfekten Ergebnissen.

Die aus dem visualisierten Decision Tree und der Feature Importance der Random Forests gewonnenen Erkenntnisse könnten von Biologen genutzt werden um Arbeitsprozesse in der Qualitätseinschätzung von Sequenzierungsdaten zu optimieren.

Die vorgeschlagenen ML-Modelle sollten in zukünftigen Versuchen mit größeren und komplexeren Datensätzen untersucht und optimiert werden, um eine erfolgsversprechende Automatisierung der Qualitätsevaluierung umsetzen zu können.

Literatur

[1] Simon Andrews. *Babraham Bioinformatics - FastQC Documentation.* URL: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/ (besucht am 30. 03. 2022).

[2] Lin Liu u. a. „Comparison of Next-Generation Sequencing Systems“. In: *Journal of Biomedicine and Biotechnology* 2012 (Juli 2012), S. 251364. ISSN: 2314-6133. DOI: 10.1155/2012/251364. URL: https://doi.org/10.1155/2012/251364.

[3] David P. Clark, Nanette Jean. Pazdernik und Michelle R. McGehee. „Next Generation Sequencing“. In: *Molecular biology*. Academic Cell, 2019.

[4] Marcos Morey u. a. „A glimpse into past, present, and future DNA sequencing“. In: *Molecular Genetics and Metabolism* 110.1 (2013). Special Issue: Diagnosis, S. 3–24. ISSN: 1096-7192. DOI: https://doi.org/10.1016/j.ymgme.2013.04.024. URL: https://www.sciencedirect.com/science/article/pii/S1096719213001546.

[5] Rute Pereira, Jorge Oliveira und Mário Sousa. „Bioinformatics and Computational Tools for Next-Generation Sequencing Analysis in Clinical Genetics“. In: *Journal of Clinical Medicine* 9.1 (2020). ISSN: 2077-0383. DOI: 10.3390/jcm9010132. URL: https://www.mdpi.com/2077-0383/9/1/132.

[6] Yu Fu u. a. „Elimination of PCR duplicates in RNA-seq and small RNA-seq using unique molecular identifiers“. In: *BMC Genomics* 19.1 (Juli 2018), S. 531. ISSN: 1471-2164. DOI: 10.1186/s12864-018-4933-1. URL: https://doi.org/10.1186/s12864-018-4933-1.

[7] Illumina. *BaseSpace Online Help: FASTQ File Format.* URL: https://support.illumina.com/help/BaseSpace_OLH_009008/Content/Source/Informatics/BS/FileFormat_FASTQ-files_swBS.htm (besucht am 30. 03. 2022).

[8] Peter J. A. Cock u. a. „The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants“. In: *Nucleic Acids Research* 38.6 (Dez. 2009), S. 1767–1771. ISSN: 0305-1048. DOI: 10.1093/nar/gkp1137. eprint: https://academic.oup.com/nar/article-pdf/38/6/1767/16769834/gkp1137.pdf. URL: https://doi.org/10.1093/nar/gkp1137.

[9] Géron Aurélien. *Hands-on machine learning with SCIKIT-learn, Keras, and tensorflow: Concepts, tools, and Techniques.* 2. Aufl. O'REILLY MEDIA, 2019.

[10] The Document Foundation. *LibreOffice Draw.* Version 2022 (7.3.1). URL: https://www.libreoffice.org/discover/draw/.

[11] William S Noble. „What is a support vector machine?“ In: *Nature Biotechnology* 24.12 (Dez. 2006), S. 1565–1567. DOI: 10.1038/nbt1206-1565.

[12] Grace Zhang. *What is the kernel trick? why is it important?* Nov. 2018. URL: https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d (besucht am 30. 03. 2022).

[13] Müller Andreas C. und Sarah Guido. *Introduction to machine learning with python: A guide for data scientists.* 2. Aufl. O'Reilly Media, Inc., 2017.

[14] Fatih Karabiber. *Gini impurity*. URL: https://www.learndatasci.com/glossary/gini-impurity/ (besucht am 30. 03. 2022).

[15] Michael L. Waskom. „seaborn: statistical data visualization“. In: *Journal of Open Source Software* 6.60 (2021), S. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[16] F. Pedregosa u. a. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

[17] J. D. Hunter. „Matplotlib: A 2D graphics environment“. In: *Computing in Science & Engineering* 9.3 (2007), S. 90–95. DOI: 10.1109/MCSE.2007.55.