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

Independent Coursework 2

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

Tabellenverzeichnis

1 Einleitung

1.1 Zielstellung

Im Independent Coursework 1 wurde gezeigt, wie FASTQ-Dateien mit vorliegender Qualitätsbewertung dazu verwendet werden können Machine Learning (ML)-Modelle zu trainieren. Dafür wurden grundlegende Statistiken und Modulstatus einer FastQC-Analyse der Dateien als Features eingesetzt.

Die vorliegende Arbeit ist eine Fortsetzung des Projekts und beschreibt die Implementierung weitergehender Features aus den Inhalten der einzelnen FastQC-Module, um zu untersuchen ob sich dadurch die Performance der ML-Modelle verbessern lässt. Der vollständige Prozess der Datenextraktion, des Feature Engineerings und der Modell-Erstellung, für das Modell mit der besten Performance, soll daraufhin in einem parametrisierten Python-Skript umgesetzt werden. Ebenso soll ein Python-Skript implementiert werden, welches auf Grundlage eines entstandenen ML-Modells eine Qualitätseinschätzung für eine neue FASTQ-Datei ausgibt. Dieses soll zudem in einem Nextflow-Prozess integriert werden, um in bestehende Pipelines eingebaut werden zu können.

1.2 Voraussetzungen

Die Implementierung des Independent Coursework 1 wird in dieser Arbeit erweitert. Daher gelten alle dort beschriebenen Software Voraussetzungen. Gleichermaßen bleiben auch die vorliegenden Daten, die Entwicklungsumgebung und das Git-Repository unverändert.

Zusätzliche Software Voraussetzungen sind Nextflow (21.10.6.5660) [1] und die Python Bibliotheken pyfastx (0.8.4) und SciPy (1.7.3) [2]. Nextflow benötigt eine Java Runtime Environment (mindestens Version 8).

Der Import zusätzlich benötigter Daten, das Feature Engineering und das Training der ML-Modelle werden weiterhin zunächst in Jupyter Notebooks umgesetzt, wobei bestehende Notebooks teilweise erweitert werden. Nachdem das bestmögliche Vorgehen etabliert wird, kann es in Python Module transferiert und durch ein Python Skript ausgeführt werden.

2 Feature Engineering

2.1 Überblick

Das Ziel des Feature Engineerings ist es die kontinuierlichen Ergebnisdaten der einzelnen FastQC-Module auf geeignete Weise in wenigen numerischen Features zu repräsentieren. Die automatische Gruppierung von Werten in einigen FastQC-Modulen ist für die vorgesehene Implementierung unerwünscht. Daher müssen diese Werte aus den rohen FASTQ-Dateien entnommen werden. Die Features für die Module werden auf unterschiedlichen Wegen gewonnen. Die Daten der meisten Module können an eine Normalverteilung, Betaverteilung oder Polynomfunktion angepasst werden und dann durch deren Parameter dargestellt werden. Für andere Module reicht eine Zusammenfassung der aus den Rohdaten gewonnen Werte. Einige Module können aufgrund des geringen Informationsgehalts für die vorliegenden Daten weggelassen werden. Die Tabelle 2.1 zeigt einen Überblick aller Module, deren Bezeichnung in der Implementierung sowie die verwendete Methode des Feature Engineerings. Im Folgenden werden die Module über die in der Tabelle angegebene Nummerierung referenziert.

FastQC Modulname	Bezeichnung	Feature Engineering Methode
Per Base Sequence Quality	module_1	Polynomfunktion aus Rohdaten
Per Tile Sequence Quality	module_2	Normalverteilung aus FastQC-Daten
Per Sequence Quality Scores	module_3	Betaverteilung aus FastQC-Daten
Per Base Sequence Content	module_4	Normalverteilung aus Rohdaten
Per Sequence GC Content	module_5	keine Repräsentation
Per Base N Content	module_6	Gesamtanteil aus Rohdaten
Sequence Length Distribution	module_7	Betaverteilung aus FastQC-Daten
Sequence Duplication Levels	module_8	Betaverteilung aus FastQC-Daten
Overrepresented Sequences	module_9	keine Repräsentation
Adapter Content	module_10	keine Repräsentation

Tabelle 2.1: Feature Engineering Methoden der FastQC-Module

2.2 Import der Rohdaten

Für das Feature Engineering der Module 1, 4 und 6 haben sich die Daten der FastQC-Ergebnisse als ungeeignet oder zu komplex herausgestellt. Um diese geeigneter darzustellen, müssen die enthaltenen Informationen, aus denen Features gewonnen werden sollen, aus einem direkten Import der FASTQ-Dateien entnommen werden. Dieser Schritt ist in dem Notebook *02_extract_raw_fastq.ipynb* implementiert. Für den Import wird die Python Bibliothek pyfastx verwendet, alternativ kann auch das Modul SeqIO der Bibliothek Biopython mit der selben Logik verwendet werden.

Beim Auslesen der Base Calls und Phred-Werte der FASTQ-Dateien werden die folgenden Werte je Position in der Sequenzierung berechnet: durchschnittlicher Phred-Wert (als Numpy-Array *means)*, Basenanteile (als Dictionary von Arrays für jede Base *base_ content*) und der gesamte Anteil an erfolglosen Base Calls (als Numpy-Array *ncontent_ total*).

Das Parsing der FASTQ-Dateien in Python ist sehr zeitaufwendig und dauert für den gesamten Datensatz von 37,7 Gigabyte ungefährt 254 Minuten. Im Vergleich dazu benötigt FastQC für die Analyse aller Daten ungefähr 47 Minuten.

Die gewonnenen Daten werden in einem Pandas Dataframe [3] für jeden Read angeordnet und als JSON-Datei exportiert. Das Dataframe für den vollständigen Datensatz im bestehenden Notebook *03_ fastqc_ data_ extraction_ and_ merge.ipynb* wird um diese Daten erweitert und exportiert.

2.3 Feature Engineering der Modulinhalte

Der vollständige Datensatz wird im Notebook *07_ feature_ engineering_ modules.ipynb* dazu verwendet die Features für die einzelnen Module zu konstruieren. Nach Extraktion der notwendigen Daten werden die entsprechenden Spalten aus dem Dataframe entfernt.

2.3.1 Modul 1

Das Feature Engineering für das erste Modul verwendet die aus den Rohdaten gewonnenen durchschnittlichen Phred-Werte. Es wird eine Polynomfunktion 4. Grades definiert und diese mittels der *curve_ fit*-Funktion aus dem SciPy Modul *optimize* an die kontinuierlichen Daten der durchschnittlichen Phred-Werte angepasst. Es konnte festgestellt werden, dass eine Polynomfunktion niedrigeren Grades keine ausreichend genaue Abbildung der Daten erzielte und eine Funktion höheren Grades kaum eine Verbesserung der Genauigkeit bewirkt.

Damit entstehen vier Features für die Repräsentation des 1. Moduls - *module_ 1_ a0*, *module_ 1_ a1*, *module_ 1_ a2* und *module_ 1_ a3*.

Das Modul beinhaltet außer der durchschnittlichen Phred-Werte auch noch deren Median und verschiedene Perzentile. Ein äquivalentes Feature Engineering für diese Daten sollte in zukünftigen Erweiterungen in Betracht gezogen werden. Für den limitierten Datensatz würde dies allerdings in zu vielen Features resultieren.

2.3.2 Modul 2

Für das Feature Engineering des zweiten Moduls werden die Qualitätsabweichungen je Kachel (engl.: Tile) gemittelt. FastQC gruppiert die einzelnen Abweichungen für jeweils fünf Positionen, mit Außnahme der ersten 9 Positionen. Die letzte Gruppe ist abhängig von der Länge des Reads und kann variabel eine Größe zwischen 1 und 4 haben. Daher wird für jeden gegebenen Wert ein Gewicht, entsprechend der Größe der Gruppe, extrahiert und damit der gewichtete Durchschnitt je Tile berechnet.

Die Ergebnisdaten besitzen eine bimodale Normalverteilung, getrennt am Nullpunkt. Daher können die Paramater der Verteilung der negativen Werte und der Verteilung der positiven Werte unabhängig voneinander berechnet werden.

Durch Untersuchung der Korrelation der entstandenen Features konnte festgestellt werden, dass es ausreicht nur die Standardabweichungen der beiden Verteilungen als Features zu verwenden. Die jeweiligen Durchschnitte werden für die weitere Betrachtung weggelassen.

Weiterhin wurde untersucht, ob der Wert der bimodalen Separation aus Formel 2.1 als Feature infrage kommt [4].

$$S = \frac{\mu_2 - \mu_1}{2\sigma_2 + 2\sigma_1} \tag{2.1}$$

Dieser zeigte jedoch in der Feature Importance des Random Forest nur einen sehr geringen Einfluss und wird in der Implementierung weggelassen.

Damit ergeben sich als Features für dieses Modul die beiden Variablen *module_2_std_pos* und *module_2_std_neg*.

2.3.3 Module 3, 7 und 8

Die Daten der Module 3, 7 und 8 bilden Häufigkeitsverteilungen ab und können einer Betaverteilung angepasst werden. Die Funktion der Betaverteilung wird auf Grundlage der Gammaverteilungsfunktion definiert. Die Daten werden wieder mittels *curve_fit* an diese Funktion angepasst. Da die Betaverteilung üblicherweise jedoch für Wahrscheinlichkeitsverteilungen ausgelegt ist, müssen die Daten zuvor auf kleiner 1 skaliert werden. Dazu werden alle Werte durch den größten Wert + 1 geteilt. Die Genauigkeit der Abbildung der Daten ist ausreichend, zeigt jedoch bei vielen der Sequenzierungsdaten Abweichungen. Es könnte sinnvoll sein in zukünftigen Arbeiten weitere Verfahren zur Zusammenfassung dieser Daten zu erforschen. Da die Ausprägungen unterschiedlicher Sequenzierungen jedoch stark voneinander abweichen können, muss hier ein Kompromiss eingegangen werden. Die resultierenden Features der drei Module sind *module_3_alpha*, *module_3_beta*, *module_7_alpha*, *module_7_beta*, *module_8_alpha* und *module_8_beta*.

2.3.4 Modul 4

Die Features für das Modul 4 werden aus den Basenanteilen je Position des Rohdaten-Imports berechnet. Durch Einsatz der Python Bibliothek NumPy werden für jede Base die Veränderung ihres Anteils von einer Position zur nächsten berechnet. Die Anteilsveränderungen aller vier Basen werden summiert.

Die Ergebnisdaten weisen eine Normalverteilung auf, womit sie sich durch den Durchschnitt und die Standardabweichung zusammenfassen lassen. Damit entstehen für das Modul 4 die beiden Features *module_4_diff_mean* und *module_4_diff_std*.

2.3.5 Modul 6

Das Modul 6 wird durch den N-Anteil der gesamten Sequenzierung zusammengefasst. Hierfür werden keine weiteren Transformationen benötigt.

2.4 Machine Learning

Die vorhandene Implementierung der ML-Modelle kann für die neuen Features übernommen werden. Das Training und die Evaluierung der Modelle für die verschiedenen Datensätze unter

Einsatz aller ursprünglichen und neuen Features befinden sich in den folgenden Notebooks:
- *08_ngs_quality_machine_learning_all_features-all-reads.ipynb*
- *08_ngs_quality_machine_learning_all_features-ecoli.ipynb*
- *08_ngs_quality_machine_learning_all_features-efcm.ipynb*
- *08_ngs_quality_machine_learning_all_features-sau.ipynb*

Zum Vergleich sind im Notebook *08_ngs_quality_machine_learning_only_new_features .ipynb* die Ergebnisse für den gesamten Datensatz ohne die Verwendung der ursprünglichen Features zu finden. Dieses erzielt eine zufriedenstellende Performance, jedoch ist der Einsatz aller Features überlegen. Das Random Forest Modell weist die beste Performance auf und wird deshalb für die folgende Pipeline eingesetzt. Obwohl die Organismus spezifischen Modelle eine bessere Performance aufweisen, soll die Pipeline auch die Möglichkeit bieten Organismus unabhängige Modelle zu produzieren, da diese einen versatileren Einsatz für Sequenzierungen bieten, die über keine zufriedenstellenden spezifischen Modelle verfügen.

3 Implementierung einer Pipeline

3.1 Python Skripte und Module

3.1.1 Überblick

Die in den Notebooks implementierte Logik wird in vier Python Module übertragen und in sinnvolle Funktionen aufgeteilt. Die Module befinden sich im *src*-Verzeichnis. Es werden drei parametrisierte Skripte zur Verfügung gestellt, welche auf die Module zugreifen und Modelle erstellen oder Evaluierungen durchführen. Die Bezeichnungen und Inhalte der Module und Skripte kann der Tabelle 3.1 entnommen werden.

Bezeichnung	Inhalt
src/fastcqc_extract.py	Import der FASTQ- und FastQC-Daten in ein Dataframe
src/data_preparation.py	Vorverarbeitung der importierten Daten
src/feature_engineering.py	Feature Engineering
src/ml_model	Training eines Modells und Evaluierung durch ein Modell
main_training.py	Trainieren eines Modells
main_single_prediction.py	Evaluierung einer FASTQ-Datei durch ein Modell
main_multiple_prediction.py	Evaluierung mehrerer FASTQ-Dateien durch ein Modell

Tabelle 3.1: Implementierte Module und Skripte

3.1.2 fastqc_extract

Das Modul *fastqc_extract* enthält alle notwendigen Schritte zum Import der notwendigen Informationen aus einer FASTQ-Datei. Jeder der Schritte ist in einer eigenen Funktion implementiert. Dabei kann jeder einzelne Import entweder aus einer Ursprungsdatei (FASTQ-Datei bzw. FastQC-Ergebnisdatei) oder aus einem vorhandenen JSON-Export durchgeführt werden. Dies ermöglicht eine erhebliche Zeitersparnis für Datensätze, deren Import schon einmal durchgeführt

wurde. Jeder erneute Import von Ursprungsdateien produziert dafür diese JSON-Export-Dateien. Die einzelnen Funktionen dieses Moduls und deren Inhalt sind in der Tabelle 3.2 beschrieben.

Funktion	Inhalt
run_fastqc_directory	führt FastQC für alle FASTQ-Dateien in einem Verzeichnis aus
import_fastqc_results *import_fastqc_results_from_source* *import_fastqc_results_from_export*	importiert die Ergebnisse der FastQC-Analyse in ein Pandas Dataframe
extract_metadata *extract_metadata_from_source* *extract_metadata_from_export*	importiert die Metadaten der Verzeichnisstruktur der FASTQ-Dateien in ein Pandas Dataframe
import_position_data *import_position_data_from_source* *extract_fastq_positions* *import_position_data_from_export*	importiert die Rohdaten der FASTQ-Dateien in ein Pandas Dataframe
import_all_reads	führt alle notwendigen Funktionen aus und fasst die einzelnen Dataframes zusammen

Tabelle 3.2: Funktionen des Moduls *fastqc_extract*

Alle Schritte werden in der vorgesehenen Reihenfolge durch die Funktion *import_all_reads* aufgerufen und die gesammelten Daten in einem Dataframe zurückgegeben. Die Parameter für den Aufruf der Funktion sind in der Tabelle 3.3 dargestellt, von hier aus werden sie an die entsprechenden Funktionen, die aufgerufen werden, weitergegeben.

Bezeichnung	Erklärung
inputdir	Verzeichnis der zu verarbeitenden FASTQ-Dateien
exportdir	Verzeichnis für vorliegende und zu erstellende Exporte
force_reimport	Import aus Ursprungsdaten erzwingen, wenn True
export	unterdrückt den JSON-Export von Zwischenergebnissen, wenn False
include_metadata	unterdrückt den Import von Metadaten, wenn False - Metadaten werden nur für das Training benötigt und müssen für die Evaluierung weggelassen werden
include_positiondata	unterdrückt den langwierigen Rohdaten-Import, wenn False
single_file	Einstellung, ob ein Verzeichnis (False) oder eine einzelne FASTQ-Datei (True) importiert werden soll

Tabelle 3.3: Parameter der Funktion *import_all_reads*

3.1.3 data_preparation

Das Modul *data_preparation* implementiert die Aufbereitung der Daten in einer Funktion *prepare_fastqc_data*. Diese verwendet als Eingabe ein Dataframe der importierten Daten. Es trans-

formiert die notwendigen Spalten des Dataframes zu numerischen Typen und extrahiert die einfachen Features aus dem Datensatz. Das aufbereitete Dataframe wird wieder zurückgegeben.

3.1.4 feature_engineering

Das Modul *feature_engineering* implementiert eine eigene Funktion für das Feature Engineering jedes Moduls. Diese werden in der Funktion *apply_feature_engineering* der Reihe nach aufgerufen und transformieren das eingegebene Dataframe in die finale Form für das ML. Das entstandene Dataframe wird als JSON-Datei exportiert und von der Funktion zurückgegeben. Die Funktion verwendet als Eingabe den aufbereiteten Datensatz *fastqc_dataset*. Die weiteren Paramater *exportdir*, *force_reimport* und *export* sind äquivalent zu den Parametern aus Tabelle 3.3.

3.1.5 ml_model

Das Modul *ml_model* stellt Funktionen für das Training der ML-Modelle, deren Performance Evaluierung, deren Export und die Klassifizierung eines Datensatzes durch ein ML-Modell bereit.

Für das Training sind zwei verschiedene Funktionen implementiert: *train_model* und *train_model_per_organism*. Erstere verwendet den vollständigen Datensatz, um ein Random Forest Modell zu trainieren. Letzteres teilt den gegebenen Datensatz in Teildatensätze je Organismus ein und erstellt durch Aufruf der ersten Funktion ein eigenes Modell für jeden. Die Funktion *train_model* verwendet das vom Feature Engineering erstellte Dataframe *fastqc_dataset* und die Parameter *exportdir*, *evaluate* und *name*. Der Parameter *exportdir* gibt das Verzeichnis an, in dem das entstandene Modell gespeichert werden soll. Der Parameter *evaluate* bietet die Option die Performance des Modells in einer Kreuzvalidierung zu evaluieren (implementiert in der Funktion *evaluate_model*). Über den Parameter *name* kann eine Bezeichnung für das zu speichernde Modell übergeben werden. Bei direktem Aufruf der Funktion wird standardmäßig „complete_dataset“ verwendet. Bei Aufruf der Funktion aus der Funktion *train_model_per_organism* wird der Name des Organismus verwendet.

Die Funktion *predict_evaluation* verwendet einen aufbereiteten Datensatz der Features *fastqc_dataset* und ein vorliegendes ML-Modell *modelpath* um eine Klassifikation (0 für „ugly“, 1 für „good“) der Qualität zurückzugeben.

3.1.6 main_training

Das Skript *main_training* implementiert einen exemplarischen Einsatz der Module um aus einem Verzeichnis von FASTQ-Dateien die ML-Modelle zu generieren. Für die Ausführung müssen drei Parameter übergeben werden: *training_data_dir* - der Pfad eines Verzeichnisses mit FASTQ-Dateien, *exports_dir* - der Pfad eines Verzeichnisses für die entstehenden JSON-Exporte und *model_dir* - der Pfad eines Verzeichnisses für die entstehenden ML-Modelle. Es werden Random Forest Modelle für den gesamten Datensatz und für die einzelnen Organismen erstellt. Deren Performance wird auch evaluiert, die Ergebnisse der Evaluierung werden im Export Verzeichnis gespeichert.

3.1.7 main_single_prediction und main_multiple_prediction

Die beiden Skripte zur Klassifikation der Qualität von FASTQ-Dateien sind nahezu identisch, das Skript *main_single_prediction* kann für Pfade einer einzelnen FASTQ-Datei verwendet werden und das Skript *main_multiple_prediction* für Pfade von Verzeichnissen mit mehreren FASTQ-Dateien.

Die Parameter, die für die Ausführung übergeben werden müssen, sind *data_dir*, *model_dir*, *exports_dir* und *organism*. Der Parameter *data_dir* ist der Pfad zu der oder den FASTQ-Dateien, *model_dir* ist der Pfad zu den vorhandenen ML-Modellen und *exports_dir* ist der Pfad für entstehende Exporte während der Datenaufbereitung sowie den Export der Ergebnisse der Klassifikation. Der Parameter *organism* dient dazu das Modell, welches verwendet werden soll, zu spezifizieren. Hier sollte also der Organismus der Sequenzierungen übergeben werden. Wenn für diesen kein spezifisches Modell vorliegt, wird das Modell für den vollständigen Datensatz verwendet.

Die Qualitätseinschätzung des ML-Modells wird in der Datei *evaluations.txt* gespeichert und in der Konsole ausgegeben. Die Datei sieht für ein Verzeichnis von FASTQ-Dateien folgendermaßen aus:

```
200709_20-07968_20-00891_S21_L000_R2_001: ugly
180525-18-3598-UW18720_S3_L001_R1_001: ugly
200707_20-07783_19-03213_S5_L000_R2_001: good
190318-19-01846-UW19617_S8_L001_R1_001: good
200707_20-07786_19-03247_S8_L000_R2_001: good
200323_20-03948_NRZ-55702_S45_L000_R2_001: ugly
191025_19-09747_UW_12546_S21_L000_R2_001: ugly
191113_19-10154_20478_S136_L000_R2_001: good
```

3.2 Einbindung in einen Nextflow Prozess

Das Python Skript *main_single_prediction* kann von einem Nextflow Prozess mit den notwendigen Parametern aufgerufen werden. Der Prozess ist in der Datei *main.nf* implementiert, die Parameter sind für eine beispielhafte Ausführung in der Datei *nextflow.config* definiert.

Der Nextflow Prozess kann innerhalb des Repositories durch den Befehl '`nextflow run main`', bzw. außerhalb des Repositories durch '`nextflow run ngs_quality_control`' ausgeführt werden. Er schreibt seine Ergebnisse in das Verzeichnis *exports_evaluation_single* und gibt den FASTQ-Dateinamen und dessen Qualitätseinschätzung in der Konsole aus.

4 Zusammenfassung

4.1 Entstandene Implementierung

Die Anforderungen des Projekts wurden in sieben Jupyter Notebooks erforscht und evaluiert. Der beste Lösungsweg, unter Verwendung der beschriebenen Features und des Random Forest Modells, wurde in einer Python Pipeline umgesetzt. Dieses setzt durch Ausführen des Skripts *main_training* den vollständigen Prozess der Datenextraktion, -aufbereitung und Modellerstellung für ein Verzeichnis von FASTQ-Dateien um. Weitere Python Skripte implementieren die Verwendung der entstandenen Modelle für die Evaluierung der Qualität neuer FASTQ-Dateien. Das Skript für die Evaluierung einer Datei wurde in einem Nextflow Prozess eingebunden und lässt sich dadurch in bestehende Nextflow Pipelines einbauen.

4.2 Fazit und Ausblick

Die Performance des Random Forest Modells mit allen erstellten Features ist sehr gut. In weiteren Arbeiten sollte erforscht werden, ob diese Performance auch für andere Datensätze besteht. Datensätze mit schlechterer Performance könnten ermöglichen, die Verbesserung von Features und Hyperparametern des Random Forest zu untersuchen.

Ein Schwachpunkt der vorliegenden Implementierung ist die besonders lange Ausführungszeit des Rohdaten-Imports. Hier sollte in folgenden Arbeiten eruiert werden, ob die Features nicht durch Features aus den FastQC-Ergebnissen ersetzt werden können. Alternativ könnte auch eine Implementierung des Imports durch performantere Systeme erkundet werden, beispielsweise durch eine eigene C++-Implementierung oder das Software-Tool fastp [5].

Weiterhin sollte erforscht werden, wie sich der erstellte Nextflow Prozess in vorhandene Pipelines integrieren lässt. Möglicherweise sind dafür noch weitere Anpassungen notwendig.

Literatur

[1] Paolo Di Tommaso u. a. „Nextflow enables reproducible computational workflows“. In: *Nature Biotechnology* 35.4 (Apr. 2017), S. 316–319. DOI: 10.1038/nbt.3820. URL: https://nextflow.io.

[2] Pauli Virtanen u. a. „SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python“. In: *Nature Methods* 17 (2020), S. 261–272. DOI: 10.1038/s41592-019-0686-2.

[3] The pandas development team. *pandas-dev/pandas: Pandas*. Version 1.4.1. Feb. 2022. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[4] Chidong Zhang, Brian E. Mapes und Brian J. Soden. „Bimodality in tropical water vapour“. In: *Quarterly Journal of the Royal Meteorological Society* 129.594 (2003), S. 2847–2866. DOI: 10.1256/qj.02.166.

[5] Shifu Chen u. a. „Fastp: An ultra-fast all-in-one FASTQ preprocessor“. In: *Bioinformatics* 34.17 (2018), S. i884–i890. DOI: 10.1093/bioinformatics/bty560.