Technische Hochschule Wildau
Technical University of Applied Sciences
WILDAU

BELEGARBEIT

In-Memory Datenbanken

WS24/25

Prof. Dr. Ralf Szymanski

Ahmed Abdelhafez
[50101618]

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	**Bedeutung**
ABC	Alphabet
DBMS	Database Management System (Datenbankmanagementsystem)
ER	Entity-Relationship
ETL	Extract, Transform, Load
LOO	Leave-One-Out
NoSQL	Not Only SQL
RAM	Random Access Memory
SQL	Structured Query Language
WS	Wintersemester

Abbildungsverzeichnis

1 Einleitung

1.1 Einleitung und Überblick

Die stetige Weiterentwicklung von datengetriebenen Technologien hat neue Chancen in der medizinischen Forschung geschaffen, vor allem in der Onkologie. Hierbei wird die Analyse von komplexen Datenstrukturen in den Vordergrund gestellt, um genauere Diagnosen und individualisierte Behandlungsansätze zu realisieren. Diese Arbeit konzentriert sich auf die Fallstudie „LaserBase“, die sich mit der Klassifizierung spektroskopischer Messdaten beschäftigt. Es soll erreicht werden, dass Tumorgewebe anhand spezifischer spektraler Eigenschaften von gesundem Gewebe unterschieden werden kann, um neue Erkenntnisse für die klinische Praxis zu gewinnen.

1.2 Herausforderungen und Fragestellungen

Die präzise Analyse von Messdaten erfordert eine umfassende Datenbereinigung und -aufbereitung. Insbesondere die Handhabung von fehlerhaften oder unvollständigen Datensätzen sowie die Gewährleistung der Reproduzierbarkeit der Ergebnisse stellen Herausforderungen dar. In Anbetracht der ansteigenden Datenmengen ist zudem die effiziente Verarbeitung ein zentraler Punkt, der durch neuartige Algorithmen und verbesserte Workflows angegangen werden muss. In Anbetracht dessen ergeben sich folgende Fragen:

- Auf welche Weise ist es möglich, Messfehler und Ausreißer sicher zu identifizieren und zu beheben
- Welche Methoden sorgen dafür, dass Daten schnell und gleichzeitig genau verarbeitet werden
- In welchem Ausmaß kann die Validität der Analyse trotz möglicher Datenabweichungen gewährleistet werden

1.3 Zielsetzung und Forschungsfragen

Diese Arbeit zielt darauf ab, ein Verfahren zu entwickeln, das eine robuste und effiziente Datenverarbeitung nutzt, um die Analyse spektroskopischer Daten zu optimieren. Im Mittelpunkt steht die Entwicklung von Methoden, um die Genauigkeit und Verlässlichkeit der Ergebnisse zu erhöhen. Die wesentlichen Forschungsfragen lauten:

Wie kann eine effiziente Analyse und Interpretation von großen Datenmengen erfolgen

Welche Algorithmen eignen sich am besten, um auf den Messdaten basierende präzise Vorhersagen zu treffen

Wie lässt sich die Verlässlichkeit der Resultate in einem klinischen Setting bestätigen.

2 Theorie zu Grundlagen und Konzepte von Datenbanksystemen

2.1 Relationale Datenbanken

Relationale Datenbanken bestehen aus Tabellen, die als Relationen bezeichnet werden und über die alle Datenzugriffe erfolgen. Neue Tabellen können einfach hinzugefügt oder gelöscht werden, wodurch Anpassungen erleichtert werden. Tabellen sind durch definierte Beziehungen verbunden, die auf mathematischen Grundlagen wie Relationenkalkül und relationaler Algebra basieren. Trotz ihrer Flexibilität weisen relationale Datenbanken Schwächen wie hohen Lese- und Schreibaufwand sowie Redundanzen auf. Bekannte Beispiele sind Oracle, MySQL und PostgreSQL.1

[1] Edwin Schicker, „Datenbanken und SQL: Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL“, 5. Auflage, Springer Vieweg, 2021.

2.2 Normalisierung

Ein wesentlicher Bestandteil der Entwicklung einer Datenbank ist die Normalisierung, deren Ziel es ist, Anomalien zu verhindern und redundante Datenbestände zu verringern. Durch die Anwendung von Normalisierungsregeln wird die Konsistenz und Effizienz der Datenbank verbessert, und zukünftige Änderungen am Datenmodell werden erleichtert.

- Die erste Normalform (1NF) zielt darauf ab, sicherzustellen, dass jede Spalte nur atomare Werte enthält, indem Wiederholungsgruppen entfernt werden.
- In der zweiten Normalform (2NF) ist es erforderlich, dass alle Attribute, die nicht vom Schlüssel abhängen, vollständig von einem Primärschlüssel abhängig sind.
- Um die Datenintegrität zu sichern, sorgt die dritte Normalform (3NF) dafür, dass zwischen Nicht-Schlüsselattributen keine transitiven Abhängigkeiten bestehen.

2.3 Datenmodellierung und ER-Diagramme

Die Datenmodellierung ermöglicht eine logische Darstellung der Struktur und Beziehungen der Daten. ER-Diagramme werden häufig genutzt, um Entitäten, Attribute und deren Beziehungen visuell abzubilden.

- Entitäten: Objekte oder Elemente, die in der Datenbank abgelegt werden (z. B. „Kunde“, „Produkt“).
- Attribute: Merkmale oder Eigenschaften einer Entität (z. B. ID, Preis, Name).
- Beziehungen: Verbindungen zwischen Entitäten (z. B. „Kunde erwirbt Artikel“).
- Kardinalitäten: Geben Sie die Anzahl der Beziehungen zwischen Entitäten an (z. B. 1:1, 1:N, N:M).2

2.4 NoSQL-Datenbanken

Um den Anforderungen zeitgemäßer Anwendungen gerecht zu werden, insbesondere im Hinblick auf große Datenmengen und unstrukturierte Daten, die über mehrere Knoten skaliert werden müssen, wurden NoSQL-Datenbanken entwickelt.

Ein zentraler Vorteil von NoSQL-Datenbanken ist ihre horizontale Skalierbarkeit, die durch das CAP-Theorem begründet wird, wobei Verfügbarkeit und Partitionstoleranz oft gegenüber Konsistenz priorisiert werden.[3]

2.5 In-Memory-Datenbanken

In-Memory-Datenbanken speichern ihre Daten im RAM statt auf Festplatten, wodurch sehr schnelle Datenzugriffe ermöglicht werden. Diese Technologie kommt vor allem in Anwendungen mit

[2] Stephan Kleuker, „Grundkurs Datenbankentwicklung: Von der Anforderungsanalyse zur komplexen Datenbankanfrage“, 5. Auflage, Springer Vieweg, 2021, S. 30.
[3] Christoph Praschl, Sebastian Pritz, Oliver Krauss, Martin Harrer, *„A Comparison Of Relational, NoSQL, and NewSQL Database Management Systems For The Persistence Of Time Series Data“, S.5.*

Echtzeitanforderungen und hohen Transaktionsvolumen zum Einsatz. In der Fallstudie LaserBase wird aufgezeigt, dass In-Memory-Datenbanken eine Echtzeitanalyse spektraler Messdaten und eine schnellere Bereitstellung von Ergebnissen ermöglichen könnten. Um große Datenmengen effizient zu verwalten, könnte die Geschwindigkeit der Verarbeitung entscheidend sein.

Nutzen von In-Memory-Datenbanken:

- Geschwindigkeit: Die Zugriffsgeschwindigkeiten sind im Vergleich zur Festplattenspeicherung um ein Vielfaches höher.
- Echtzeitfähigkeit: Optimal für Anwendungen, die sofortige Resultate benötigen, wie Finanztransaktionen oder Analysen.
- Skalierbarkeit: Zeitgemäße Systeme bieten Optionen zur effizienten Verwaltung großer Datenmengen im RAM.

2.6 Optimierungsstrategien für In-Memory-Datenbanken

Um die Leistung und Effizienz zu maximieren, ist es entscheidend, In-Memory-Datenbanken zu optimieren. Weil diese Systeme vollständig im RAM arbeiten, braucht es spezialisierte Strategien für ihre Verwaltung, damit die Speicherressourcen optimal genutzt werden und zugleich eine hohe Verfügbarkeit gewährleistet ist.

Eine effiziente Verwaltung des Speichers ist hierbei entscheidend, da sie die Verwendung der begrenzten Ressourcen optimiert und die Echtzeitfähigkeiten der Datenbanken gewährleistet. Die Fragmentierung kann verringert und der Speicher dynamisch an häufig verwendete Datengrößen angepasst werden, indem Methoden wie die Speicherpools-Verteilung angewendet werden. Darüber hinaus steigern Algorithmen zur Komprimierung und Zusammenführung freier Speicherbereiche die Verfügbarkeit größerer Speicherblöcke für zukünftige Anforderungen. Ein weiteres bedeutendes Vorgehen stellt die dynamische Anpassung der Speicherpool-Verwaltung dar, bei der Blockgrößen flexibel auf Basis statistischer Daten zu Speicheranforderungen verändert werden. Um die hohe Performance und Skalierbarkeit von In-Memory-Datenbanken auch bei zunehmenden Belastungen zu bewahren, sind derartige Optimierungsansätze unerlässlich. [4]

3 Fallstudie Laserbase

Die Implementierung umfasst die Bereinigung und Aggregation der Referenzdaten sowie den Vergleich mit den Testdaten. Dabei werden effiziente Algorithmen und SQL-basierte Optimierungen genutzt, um eine präzise und performante Analyse sicherzustellen.

[4] Liu Feng, Zhang Kun, *„A Novel Real-Time Database Memory Management Approach“*, 2nd International Conference on Industrial Mechatronics and Automation (ICIMA), 2010, S.417-420.

3.1 Datenaufbereitung

- Laden und Bereinigung der Referenzdaten

(Python-Funktion zur Datenbereinigung und Durchschnittsberechnung kein Datum)

```
[2]: def lade_referenzdaten(dateipfad):
        """
        Lädt die Referenzdaten aus einer CSV-Datei, bereinigt sie und berechnet den Durchschnitt pro Gruppe.
        """
        daten = pd.read_csv(dateipfad, sep=';', skipinitialspace=True, decimal=',')
        daten.columns = daten.columns.str.strip()

        # Debugging: Tatsächliche Spaltennamen anzeigen
        print("🔍 Tatsächliche Spaltennamen in der Datei:", daten.columns.tolist())

        erwartete_spalten = ['Lfd. Nr.', 'Patient', 'Befall', 'Position', 'Art', 'Versuch', 'Wave', 'Value']
        if not all(spalte in daten.columns for spalte in erwartete_spalten):
            raise ValueError(f"❌ Die Datei {dateipfad} enthält nicht die erwarteten Spalten: {erwartete_spalten}")

        # Filter auf Gewebe
        daten = daten[daten['Art'] == 'Gewebe']

        # Werte bereinigen und konvertieren
        if daten['Value'].dtype == 'object':
            daten['Value'] = daten['Value'].str.rstrip(',').str.replace(',', '.').astype(float)

        daten = daten.dropna(subset=['Value'])
        daten['Wave'] = daten['Wave'].astype(str).str.replace(',', '.').astype(float).round(4)

        # Durchschnittswerte mit LOO berechnen
        daten['Average'] = daten.groupby(['Patient', 'Position', 'Wave', 'Befall'])['Value'].transform(
            lambda gruppe: berechne_mittelwert_loo(gruppe)
        )

        return daten[['Patient', 'Position', 'Wave', 'Average', 'Befall']].drop_duplicates().dropna(subset=['Average'])
```

(*Abbildung 1: Python-Funktion zur Datenbereinigung und Durchschnittsberechnung*)

1. Die Funktion liest die Referenzdaten aus einer CSV-Datei ein.
2. Es wird überprüft, ob die Datei alle erforderlichen Spalten enthält.
3. Es werden nur Daten mit Art = 'Gewebe' weiterverarbeitet.
4. Falls Value als Text gespeichert ist, wird es in eine float-Zahl umgewandelt.
5. Fehlende Werte (NaN) werden entfernt, und Wave wird gerundet.
6. Der Durchschnittswert wird für jede Kombination aus Patient, Position, Wave, Befall berechnet.

- Berechnung des Mittelwerts mit Leave-One-Out (LOO)

```python
[3]: def berechne_mittelwert_loo(werte):
    """
    Berechnet den Durchschnitt einer Gruppe unter Verwendung der Leave-One-Out-Methode (LOO).
    """
    werte = np.array(werte)
    positive_werte = werte[werte >= 0]

    # Debugging: Zeige die Werte
    print(f"🔍 Werte in der Gruppe: {positive_werte}")

    if len(positive_werte) == 0:
        print("⚠️ Keine positiven Werte gefunden.")
        return np.nan

    mittelwert = positive_werte.mean()

    # Verhindere Division durch Null oder NaN
    if mittelwert == 0 or np.isnan(mittelwert):
        print("⚠️ Mittelwert ist 0 oder NaN:", mittelwert)
        return np.nan

    abweichung = np.abs((positive_werte - mittelwert) / mittelwert)
     # Debugging: Zeige Abweichungen
    print(f"🔍 Abweichungen: {abweichung}")

    # Entferne Ausreißer
    if np.sum(abweichung > 0.3) > 0.3 * len(positive_werte):
        positive_werte = positive_werte[abweichung <= 0.3]

    return positive_werte.mean() if len(positive_werte) > 0 else mittelwert
```

(*Abbildung 2: Python-Funktion zur Berechnung des Durchschnitts mit der Leave-One-Out-Methode (LOO)*)

1. Die Funktion berechnet den Durchschnitt einer Gruppe mit der Leave-One-Out-Methode (LOO).
2. Zuerst werden alle negativen Werte entfernt.
3. Falls keine positiven Werte vorhanden sind, wird NaN zurückgegeben.
4. Der Mittelwert der Werte wird berechnet.
5. Falls der Mittelwert 0 oder NaN ist, wird NaN zurückgegeben.
6. Es wird überprüft, ob mehr als 30% der Werte stark abweichen, falls ja, werden diese entfernt.
7. Der finale Mittelwert wird berechnet und zurückgegeben.

- Laden und Bereinigung der Testdaten

```python
[4]: def lade_testdaten(dateipfad):
    """
    Lädt und bereinigt die Testdaten aus einer CSV-Datei.
    """
    daten = pd.read_csv(dateipfad, sep=';', skipinitialspace=True, decimal=',')
    daten.columns = daten.columns.str.strip()

    # Debugging: Tatsächliche Spaltennamen anzeigen
    print("🔍 Tatsächliche Spaltennamen in der Datei:", daten.columns.tolist())

    erwartete_spalten = ['wave', 'value']
    if not all(spalte in daten.columns for spalte in erwartete_spalten):
        raise ValueError(f"❌ Die Datei {dateipfad} enthält nicht die erwarteten Spalten: {erwartete_spalten}")

    if daten['wave'].dtype == 'object':
        daten['wave'] = daten['wave'].str.replace(',', '.').astype(float)

    daten['wave'] = daten['wave'].round(4)

    return daten[['wave', 'value']]
```

(*Abbildung 3: Python-Funktion zum Laden und Bereinigen von Testdaten aus einer CSV-Datei)*

1. Die Funktion lädt und bereinigt die Testdaten.
2. Es wird überprüft, ob die Datei die Spalten wave und value enthält.
3. Falls wave als Text gespeichert ist, wird es in eine float-Zahl umgewandelt.

4. wave-Werte werden gerundet, um Präzisionsfehler zu minimieren.

3.2 Tumortests

- Analyse und Bewertung:
  Zur Identifizierung von Tumorerkrankungen, die auf spektralen Messwerten beruhen, kommen spezielle Testmethoden zum Einsatz. Mit diesen Untersuchungen wird geprüft, ob es eine Korrelation zwischen bestimmten Wellenlängenmustern und gesundem bzw. tumorbefallendem Gewebe gibt. Messwerte aus verschiedenen Testproben werden mit bekannten Referenzwerten verglichen, um dies zu erreichen.

  Ziel ist es, für jede Testprobe eine genaue Beurteilung vorzunehmen, ob sie als tumorfrei (N) oder tumorbefallen (T) eingestuft werden kann. Um auch große Datenmengen bewältigen zu können, muss der Prozess effizient gestaltet werden. Die Umsetzung geschieht im Arbeitsspeicher, um rasche Berechnungen zu gestatten. Zur Bestimmung des endgültigen Testergebnisses kommt eine Schwellenwertanalyse zum Einsatz.
- Vergleichungsprozesse:
  Die Funktion vergleiche_testdaten() filtert die Referenzdaten anhand des angegebenen Befall-Filters, sodass nur relevante Einträge berücksichtigt werden. Danach werden die Test- und Referenzdaten nach der Wellenlänge sortiert, um eine effiziente Zuordnung zu gewährleisten. Anschließend erfolgt ein Abgleich der Testwerte mit den nächstgelegenen Referenzwerten, wobei eine Toleranz von 0,5 berücksichtigt wird. Zum Schluss folgt die Klassifizierung: Ein Testwert wird als „N“ (tumorfrei) gekennzeichnet, wenn er zwischen 50 % und 150 % des Referenzwertes liegt, andernfalls als „T“ (tumorbefallen).

```python
def vergleiche_testdaten(referenzdaten, testdaten, befall_filter):
    """
    Vergleicht Testdaten mit den Referenzdaten nach einem bestimmten Befall-Filter.
    """
    referenzdaten['Befall'] = referenzdaten['Befall'].astype(str).str.strip()
    gefilterte_referenzdaten = referenzdaten[referenzdaten['Befall'] == befall_filter].copy()

    gefilterte_referenzdaten.sort_values('Wave', inplace=True)
    testdaten.sort_values('wave', inplace=True)

    zusammengeführt = pd.merge_asof(
        testdaten, gefilterte_referenzdaten,
        left_on='wave', right_on='Wave',
        direction='nearest', tolerance=0.5
    )

    zusammengeführt['befall-ergebnis'] = zusammengeführt.apply(
        lambda row: "N" if row['Average'] / 1.5 <= row['value'] <= row['Average'] * 1.5 else "T",
        axis=1
    )

    return zusammengeführt.dropna(subset=['Average'])
```

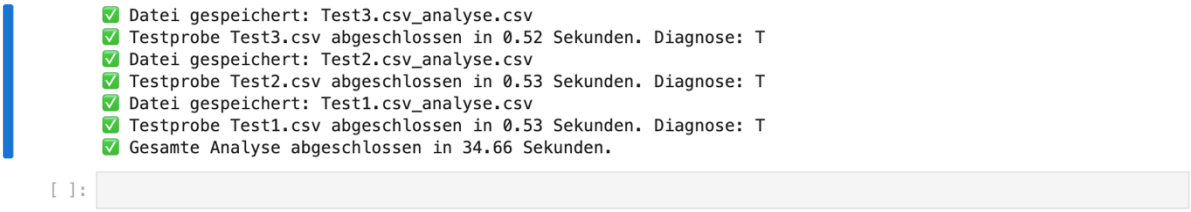
(*Abbildung 4: Python-Funktion zum Vergleich von Testdaten mit Referenzdaten*)

Die dargestellte Implementierung beinhaltet zwei wesentliche Funktionen zur Analyse von Tumortests. Die Funktion ermittle_wellenlänge_status legt den Status jeder Wellenlänge fest, indem sie die Daten nach Wave gruppiert und den Anteil der „N“-Ergebnisse in jeder Gruppe ermittelt. Übersteigt dieser Anteil einen festgelegten Schwellenwert (standardmäßig 70 %), wird die Wellenlänge als „N“ (tumorfrei) klassifiziert, andernfalls als „T“ (Tumorbefall). Die Funktion analysiere_testprobe bearbeitet eine einzelne Testdatei, indem sie zunächst einen Vergleich der Testdaten mit den Referenzdaten vornimmt und die Wellenlängenstatus bestimmt. Daraufhin wird das abschließende Tumorergebnis ermittelt, wobei der Schwellenwert für „N“-Ergebnisse zugrunde gelegt wird. Die Resultate werden in einer CSV-Datei abgelegt, und die Verarbeitungsdauer wird ausgegeben. Dank dieser mehrstufigen Verarbeitung können große Testdatensätze effizient analysiert und Tumorstatus klar klassifiziert werden.

```python
def ermittle_wellenlänge_status(daten, schwellenwert=0.7):
    """
    Bestimmt den finalen Status pro Wellenlänge basierend auf der Anzahl von 'N'.
    """
    gruppiert = daten.groupby('Wave')
    ergebnisse = []

    for wave, gruppe in gruppiert:
        n_anzahl = gruppe['befall-ergebnis'].value_counts().get('N', 0)
        gesamtanzahl = len(gruppe)
        ergebnis = "N" if n_anzahl / gesamtanzahl >= schwellenwert else "T"
        ergebnisse.append({'wave': wave, 'ergebnis': ergebnis})

    return pd.DataFrame(ergebnisse)

def analysiere_testprobe(testdatei, testdaten, referenzdaten):
    """
    Analysiert eine einzelne Testprobe und bestimmt das Tumor-Ergebnis.
    """
    start_zeit = time.time()

    ergebnis_n = vergleiche_testdaten(referenzdaten, testdaten, 'N')
    ergebnis_df = ermittle_wellenlänge_status(ergebnis_n)

    testdaten_mit_ergebnis = testdaten.merge(ergebnis_df, on='wave', how='left')

    finale_diagnose = "N" if ergebnis_df['ergebnis'].value_counts().get('N', 0) / len(ergebnis_df) >= 0.7 else "T"

    speichere_csv(testdaten_mit_ergebnis, f"{testdatei}_analyse.csv")

    dauer = time.time() - start_zeit
    print(f"✅ Testprobe {testdatei} abgeschlossen in {dauer:.2f} Sekunden. Diagnose: {finale_diagnose}")
```

(*Abbildung 5: Python-Funktionen zur Analyse von Testproben und Bestimmung des finalen Status basierend auf Schwellenwerten*)

Die kommende gezeigte Ausgabe bestätigt die erfolgreiche Durchführung der Tumortests. Die Analyse umfasste drei Testproben (Test1.csv, Test2.csv und Test3.csv), von denen jede innerhalb von 0.52 bis 0.53 Sekunden verarbeitet wurde.
Das Ergebnis jeder Testprobe wurde klassifiziert, wobei alle Proben das Ergebnis "T" (Tumorbefall) erhielten. Die vollständigen Analyseergebnisse wurden in separaten CSV-Dateien gespeichert (z. B. Test1.csv_analyse.csv). Die gesamte Analyse, einschließlich des Ladens der Daten, der Verarbeitung und der Speicherung der Ergebnisse, wurde in 34.66 Sekunden abgeschlossen.



(Abbildung 6: Ergebnisse der Analyse von Testproben mit Laufzeit und Diagnose)

3.3 Relationale Datenbank Abfragen

Um Test- und Referenzdaten strukturiert zu speichern und abzufragen, werden relationale Datenbanken verwendet. Dadurch wird eine effiziente Analyse großer Datenmengen möglich. SQL-Optimierungen ermöglichen es, Berechnungen direkt in der Datenbank vorzunehmen, was zu einer erheblichen Steigerung der Verarbeitungsgeschwindigkeit führt. Hier wird die Struktur der SQL-Abfragen beschrieben, die zur Klassifikation der Tumordaten verwendet werden.

- Berechnung aggregierter Referenzdaten:

  - ReferenzGruppiert: Berechnet für jede Gruppe (Patient, Position, Wellenlänge, Befall) den Mittelwert, die Anzahl der Werte und die Summe der Quadrate der Referenzwerte.
  - BerechneteVarianz: Berechnet die Varianz, die später zur Identifikation von Ausreißern genutzt wird.
  - GefilterteReferenzen: Berechnet die Standardabweichung für jede Gruppe.
  - BereinigteReferenzen: Entfernt Ausreißer (Werte außerhalb von zwei Standardabweichungen).
  - LEFT JOIN der Testdaten mit den bereinigten Referenzwerten.
  - Toleranz von 0.5 für den Wellenlängen-Vergleich.
  - Klassifikation der Testwerte

- Innerhalb des Bereichs 50 % – 150 % des Referenzwerts → "N" (Tumorfrei).
- Außerhalb dieses Bereichs → "T" (Tumorbefallen).

```
# SQL-Abfrage zur Verarbeitung der Referenzdaten
sql_verarbeitung = """
    WITH ReferenzGruppiert AS (
        SELECT
            Patient, Position, Wave, Befall,
            AVG(Value) AS Mittelwert, COUNT(Value) AS Anzahl,
            SUM(Value * Value) AS SummeQuadrate
        FROM ReferenzDaten
        WHERE Art = 'Gewebe' AND Value >= 0
        GROUP BY Patient, Position, Wave, Befall
    ),
    BerechneteVarianz AS (
        SELECT
            Patient, Position, Wave, Befall, Mittelwert,
            (SummeQuadrate - Anzahl * Mittelwert * Mittelwert) / NULLIF(Anzahl, 0) AS Varianz
        FROM ReferenzGruppiert
    ),
    GefilterteReferenzen AS (
        SELECT
            Patient, Position, Wave, Befall, Mittelwert,
            SQRT(Varianz) AS Standardabweichung
        FROM BerechneteVarianz
        WHERE Varianz IS NOT NULL
    ),
    BereinigteReferenzen AS (
        SELECT
            r.Patient, r.Position, r.Wave, r.Befall,
            AVG(r.Value) AS Durchschnittswert
        FROM ReferenzDaten r
        INNER JOIN GefilterteReferenzen f
        ON r.Patient = f.Patient
           AND r.Position = f.Position
           AND r.Wave = f.Wave
           AND r.Befall = f.Befall
        WHERE ABS(r.Value - f.Mittelwert) <= 2 * f.Standardabweichung
        GROUP BY r.Patient, r.Position, r.Wave, r.Befall
    )
    SELECT
        t.wave AS TestWave,
        r.Wave AS RefWave,
        r.Durchschnittswert,
        t.value,
        CASE
            WHEN t.value BETWEEN r.Durchschnittswert * 0.5 AND r.Durchschnittswert * 1.5 THEN 'N'
            ELSE 'T'
        END AS BefallErgebnis
    FROM PruefungsDaten t
    LEFT JOIN BereinigteReferenzen r
    ON ABS(t.wave - r.Wave) <= 0.5;
"""
```

(Abbildung 7: SQL-Abfrage zur Verarbeitung von Referenzdaten und Berechnung von Durchschnittswerten sowie Standardabweichungen)

- AggregierteErgebnisse:
    - Zählt die Anzahl der "N"-Klassifikationen pro Wellenlänge
    - Ermittelt den Gesamtwert der Testwerte pro Wellenlänge.
    - Wenn Endergebnis mehr als 70 % der Werte als "N" klassifiziert wurden → "N" (Tumorfrei) Andernfalls "T" (Tumorbefallen).

```
# SQL-Abfrage für aggregierte Ergebnisse
sql_aggregierte_ergebnisse = """
    WITH AggregierteErgebnisse AS (
        SELECT
            TestWave,
            COUNT(CASE WHEN BefallErgebnis = 'N' THEN 1 ELSE NULL END) AS N_Anzahl,
            COUNT(*) AS GesamtAnzahl
        FROM (
            SELECT
                t.wave AS TestWave,
                r.Wave AS RefWave,
                r.Durchschnittswert,
                t.value,
                CASE
                    WHEN t.value BETWEEN r.Durchschnittswert * 0.5 AND r.Durchschnittswert * 1.5 THEN 'N'
                    ELSE 'T'
                END AS BefallErgebnis
            FROM PruefungsDaten t
            LEFT JOIN BereinigteReferenzen r
            ON ABS(t.wave - r.Wave) <= 0.5
        )
        GROUP BY TestWave
    )
    SELECT
        TestWave,
        N_Anzahl * 1.0 / GesamtAnzahl AS N_Prozent,
        CASE
            WHEN N_Anzahl * 1.0 / GesamtAnzahl >= 0.7 THEN 'N'
            ELSE 'T'
        END AS Endergebnis
    FROM AggregierteErgebnisse;
"""
final_ergebnisse = pd.read_sql(sql_aggregierte_ergebnisse, verbindung)

# Aggregierte Ergebnisse speichern
final_ergebnisse.to_csv(f"{test_datei}_finale_ergebnisse.csv", sep=";", decimal=",", index=False)

dauer = time.time() - start_zeit
print(f"Analyse von {test_datei} abgeschlossen in {dauer:.2f} Sekunden.")
```

(Abbildung 8: SQL-Abfrage zur Aggregation von Ergebnissen und Bestimmung finaler Testergebnisse basierend auf Schwellenwerten)

- Erstellung von Indizes für Performance-Optimierung:
    - Index auf Wellenlänge ("Wave") → Beschleunigt Abfragen mit Wellenlängen-Vergleich.
    - Index auf Patient und Position → Verbessert Leistung beim Gruppen-Vergleich.

```
# Indizes erstellen, um die SQL-Abfragen zu optimieren
verbindung.execute("CREATE INDEX idx_wave ON ReferenzDaten(Wave);")
verbindung.execute("CREATE INDEX idx_patient_position ON ReferenzDaten(Patient, Position);")
```

(Abbildung 9: Erstellung von Indizes zur Optimierung von SQL-Abfragen)

3.4 Diskussion

Die Tumortests, die in den Abschnitten 3.2 und 3.3 durchgeführt werden, beruhen auf zwei verschiedenen methodischen Herangehensweisen: der In-Memory-Verarbeitung mit Pandas sowie der Verarbeitung über relationale Datenbanken mit SQL. Jede der beiden Methoden weist spezifische Vor- und Nachteile auf, die sich vor allem hinsichtlich der Datenmenge, Performance und Skalierbarkeit unterscheiden.

Bei der In-Memory-Verarbeitung kommen Pandas zum Einsatz, um die Referenzdaten zu laden und zu bereinigen sowie Durchschnittswerte mit der Leave-One-Out (LOO)-Methode zu ermitteln. Testwerte werden mit den ermittelten Durchschnittswerten verglichen und klassifiziert als „N“ (tumorfrei) oder „T“ (tumorbefallen), wenn sie innerhalb eines akzeptablen Bereichs von 50 % bis 150 % des Referenzwerts liegen. Die Verarbeitung findet unmittelbar im Hauptspeicher statt, was schnelle Berechnungen ermöglicht, solange die Datenmenge begrenzt ist.
Im Unterschied dazu verwendet die SQL-basierte Methode eine relationale Datenbank, um große Mengen von Test- und Referenzdaten zu verarbeiten und zu speichern. Statistische Verfahren, vor allem die Berechnung der Standardabweichung zur Identifikation und Eliminierung von Ausreißern, ermöglichen eine effiziente Bereinigung von SQL-Abfragen. Bei großen Datenmengen kann SQL durch indizierte Abfragen Optimierungen vornehmen. Zudem erlaubt es eine aggregierte Bewertung, die zu einer fundierteren Entscheidungsfindung beiträgt. Eine bedeutende Erweiterung in SQL ist die Schwellenwertberechnung, die das Verhältnis von „N“-Ergebnissen pro Wellenlänge bestimmt. Wurden mindestens 70 % der Werte einer Wellenlänge mit „N“ klassifiziert, so gilt die gesamte Testprobe als tumorfrei.

Der Datenverarbeitung und der Entscheidungsfindung kommt ein wesentliche Differenz zwischen den beiden Herangehensweisen zu:

- Pandas führt Berechnungen einzeln pro Testwert durch, was bei kleineren Datensätzen schnell ist, aber keine übergreifende Entscheidungslogik enthält.
- SQL hingegen aggregiert die Ergebnisse und bestimmt ein Gesamtresultat basierend auf einer statistischen Analyse über mehrere Testwerte hinweg.

Pandas eignet sich gut für explorative Analysen und kleine bis mittelgroße Datensätze, während SQL die bessere Wahl für skalierbare, robuste und standardisierte Berechnungen ist. Insbesondere für zukünftige Szenarien mit zehntausenden von Testproben wird eine relationale Datenbank mit optimierten SQL-Abfragen empfohlen, da sie eine konsistente, automatisierte und speicherfreundliche Lösung bietet.

Eine optimale Herangehensweise könnte darin bestehen, beide Methoden zu kombinieren: Pandas für die Vorverarbeitung und schnelle Analysen zu nutzen, während SQL für die endgültige Speicherung und Auswertung der Testergebnisse verwendet wird. Mit dieser hybriden Strategie würden die Flexibilität der In-Memory-Verarbeitung sowie die Skalierbarkeit und Performance von SQL kombiniert werden.

4 Zusammenfassung

Diese Belegarbeit untersucht die Verwendung moderner In-Memory-Datenbanklösungen zur Analyse umfangreicher medizinischer Datensätze. Die Arbeit ist in drei Hauptabschnitte gegliedert:

1. Einleitung und Zielsetzung:
   Die Einführung hebt die Herausforderungen bei der Verarbeitung großer Datensätze hervor, insbesondere in der medizinischen Diagnostik, wo Geschwindigkeit und Genauigkeit entscheidend sind. Die Zielsetzung ist die Entwicklung eines Systems, das Tumordiagnosen basierend auf spektralen Daten optimiert und skaliert.
2. Grundlagen:
   Dieser Abschnitt bietet eine theoretische Grundlage zu den Konzepten relationaler und NoSQL-Datenbanken sowie den Vorteilen von In-Memory-Datenbanken. Besonderer Fokus liegt auf der Normalisierung, Datenmodellierung und Optimierungstechniken, um eine effiziente Speicherung und Abfrage von Daten zu gewährleisten.
3. Fallstudie Laserbase:
   - Datenaufbereitung: In diesem Abschnitt wird die Bereinigung und Transformation der Daten beschrieben, wobei Leave-One-Out-Methoden und Durchschnittswertberechnungen für eine genauere Analyse zum Einsatz kommen.
   - Tumortests: In dem Abschnitt wird beschrieben, wie Tests mit den bereinigten Daten durchgeführt werden, wobei Algorithmen zum effizienten Klassifizieren von Tumorproben implementiert wurden.
   - Relationale Datenbankabfragen: Um die Performance zu steigern und schnelle Ergebnisse zu liefern, wurden mit SQL-Abfragen fortschrittliche Berechnungen und Aggregationen direkt in einer relationalen In-Memory-Datenbank durchgeführt.
   - Diskussion: Eine eingehende Betrachtung der beiden Ansätze offenbart die Vorzüge und Nachteile sowohl der Pandas- als auch der SQL-basierten Lösungen.

Zusammenfassung:
Die Arbeit zeigt, dass die Kombination von In-Memory-Datenbanken und optimierten Abfragen erhebliche Vorteile für die Tumordiagnostik bietet. SQL-Abfragen bieten Skalierbarkeit und Präzision, während die parallele Verarbeitung durch Pandas hohe Geschwindigkeit gewährleistet. Die Resultate der Fallstudie Laserbase unterstreichen die Bedeutung effizienter Datenverarbeitungstechniken in der medizinischen Forschung.

5 Literaturverzeichnis

Schicker, E. (2021). Datenbanken und SQL: Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL (5. Auflage, ISBN 978-3-658-16128-6). Springer Vieweg. Fakultät Informatik und Mathematik, OTH Regensburg. DOI: 10.1007/978-3-658-16129-3.

Kleuker, S. (2021). Grundkurs Datenbankentwicklung: Von der Anforderungsanalyse zur komplexen Datenbankanfrage (5. Auflage, ISBN 978-3-658-43022-1). Springer Vieweg. Hochschule Osnabrück, Fakultät Ingenieurwissenschaften und Informatik. DOI: 10.1007/978-3-658-43023-8.

Praschl, C., Pritz, S., Krauss, O., & Harrer, M. (2021). A Comparison Of Relational, NoSQL, and NewSQL Database Management Systems For The Persistence Of Time Series Data, S. 5.

Feng, L., & Kun, Z. (2010). A Novel Real-Time Database Memory Management Approach. 2nd International Conference on Industrial Mechatronics and Automation (ICIMA), 417-420.

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.

Die aus fremden Werken wörtlich oder sinngemäß übernommenen Gedanken sind unter Angabe der Quellen gekennzeichnet.

Berlin, 02.02.2025 ..... Ahmed Abdelhafez

Ort, Datum Unterschrifft