Seminar-Vortrag „Vergleich von Data-Mining-Verfahren zur Outlier Detection“ - Skript

*Verbesserungsvorschläge*

* *Ein Distanz- (evtl. eine aus {Control Chart, Linear Regression, Manhattan Distance Techniques} (Q3)) und ein dichtebasiertes Verfahren, diese vergleichen. Nicht gut und schlecht, sondern nach Anwendungsfall.*
* *Inhalt u. a.:* *Einführung zu Outlier-Detection-Verfahren, grobe Kategorien, ein Beispiel*
* ***Vergleichskriterium****, z. B. ob oder wie nominal*

Einführung zu Outlier-Detection-Verfahren

Anwendungsgebiete für Outlier-Detection (3):

* Betrugserkennung bei Kreditkarten, Handynutzung oder auch Staatshilfeleistungen
* Vorbeugung vor Kreditbetrug oder potenziell problematischen Kunden
* Angriffserkennung bei Computer-Netzen
* Fehlerdiagnose bei Maschinen (wie z.B. Motoren, Pipelines)
* Erkennung struktureller Defekte, z.B. rissige Stahlträger in der Produktion in einer Fabrik
* Überwachung des Gesundheitszustandes (z.B. Herzfrequenzmesser)
* Erkennung ungewöhnlicher Datenbankeneinträge

# Grobe Kategorien

Typen von Verfahren zur Ausreißer-Erkennung, u.a. (5, 6):

* Dichte-basiert
  + Die Dichte (~Punkte pro Raum) wird zur Ausreißerbestimmung herangezogen.
  + Z.B. **DBSCAN**, **LOF** (=>“Local Outlier Factor”)
* Statistisch-basiert
* Entfernungs-basiert
  + Z.B. **ORCA**
* Cluster-basiert
* Klassifikations-basiert
* Modell-basiert
  + Z.B. **Isolation Forest**

# Ein dichte-basiertes Verfahren: DBSCAN

* “Density-Based Spatial Clustering of Applications with Noise”
* Ausreißer sind diejenigen Punkte, die in einem Radius von epsilon (**ε-Umgebung**) weniger als p (**Dichte**) Nachbarn haben
* **ε-Umgebung**: Der Bereich um einen Punkt p, in dem andere Punkte maximal ε weit von p entfernt sind.
* **minPts**: Eine festzulegende Anzahl an Punkten, die eine Mindestdichte beschreibt
* **Kernpunkt („Core Point“)**: Ein Datenpunkt gilt als Kernpunkt, wenn sich in seiner ε-Umgebung, mindestens minPts Datenpunkte (inkl. sich selbst) befinden.
* **Direkt-dichte-erreichbar („Direct Density Reachable“)**: Ein Punkt q ist von einem Punkt p genau dann direkt-dichte-erreichbar, wenn p Kernpunkt ist und q in dessen ε-Umgebung liegt (eine nicht-symmetrische Eigenschaft).
* **Dichte-erreichbar („Density Reachable“)**: Ein Punkt q ist von einem Punkt p dichte-erreichbar, wenn zwischen p und q Punkte liegen, die, von p nach q führend, der Reihe nach direkt-dichte-erreichbar sind.
* **Dichte-verbunden („Density Connected“)**: Zwei Punkte p und q sind dichte-verbunden, wenn sie von einem dritten Punkt o dichte-erreichbar sind.
* **Randpunkt („Border Point“)**: Ein Punkt, der kein Kernpunkt ist, aber von einem Kernpunkt aus direkt-dichte-erreichbar ist.
* **Rauschpunkt („Noise“)**: Ein Punkt, der weder Kern- noch Randpunkt ist.
* OPTICS, LSDBC und HDBSCAN (Border Points gehören hierbei nicht automatisch zu einem Cluster) sind DBSCAN-Varianten, die Gruppen auf verschiedenen Hierarchie-Ebenen finden können
* Passende Werte für die Parameter Epsilon und MinPts mithilfe einer Heuristik bestimmen (7):
  + minPts = 4, da größere Werte bei wesentlich höherem Rechenaufwand kaum einen Unterschied machen
  + Berechnen und zuordnen der Distanz eines jeden Punktes zu seinem viert-nächsten Nachbarn
  + Die Punkte nach absteigenden Entfernungen sortieren und graphisch darstellen
  + (Siehe Bild) Der erste Punkt, ab dem die Kurve schlagartig abflacht wird ausgewählt und dessen Distanz als Epsilon festgelegt.
* Fakten zum Algorithmus:
  + Nachbarschaftssuchen werden nur für „undefined“-Punkte durchgeführt
  + Jeder Punkt, für den eine Nachbarschaftssuche durchgeführt wird, wird anschließend entweder einem Cluster zugeordnet oder als Rauschpunkt markiert
  + Die einzige Möglichkeit einer Neuzuordnung eines Punktes ist, wenn dieser von einem Rauschpunkt zu einem Cluster-zugehörigen Punkt wird (Zeile 11)

# Ein Modell-basiertes Verfahren: Isolation Forest

* Ein Isolation Forest besteht aus Isolation Trees und letztere sind Bäume, die Datenpunkte anhand der **Werte** ihrer **Attribute** voneinander isolieren.
* Die Auswahl der **Attribute** und des **Split-Werts** geschieht **zufällig**. Letzterer befindet sich **innerhalb der Reichweite** der vorhandenen Werte.
* Die **Datenpunkte** bilden die **Blätter** des Baumes.
* **Ausreißer** haben, im Vergleich zu normalen Datenpunkten, eine **kürzere Pfadlänge**.
* Nicht nur einen Baum, sondern **einen ganzen Wald**, um hier **Ausreißer** bezüglich Pfadlängen mithilfe des **Gesetzes der großen Zahlen** auszugleichen (Bild: Annäherung jeweils an **4.02** und **12.82**)
* Zwei Phasen: Aufbau des Waldes und Auswertung.
* Zwei Parameter nötig:
  + Teilprobengröße ψ
    - Anhand dieser Teilmenge an Datenpunkten wird der Wald aufgebaut
    - Dies reduziert die Laufzeit und die Speicherkomplexität, ohne die Ausreißererkennung merklich zu verschlechtern.
    - Dies verringert **Maskierungs- (=>Falsch-Negativ)** und **Überflutungseffekt (=>Falsch-Positive)**
    - **Baumgrößenlimit** l wird damit wie folgt berechnet: **l = ceiling(log2ψ)**
    - Hilfreicher Wert, da fein genug: ψ = 28 = 256
  + Baumanzahl t
    - Hilfreicher Wert, da Konvergenz meist schon bei weit geringeren Anzahlen erfolgt: t = 100
* Der **Anomaly Score** („Anomaliewert“) s beschreibt die **Wahrscheinlichkeit**, dass ein Punkt x ein Ausreißer ist. Die Formel lautet: s(x, n) = 2- E(h(x)) / c(n)
  + s ∈ (0, 1], für h(x) ∈ (0, n-1]
  + Werte nah an **1** deuten auf einen **Ausreißer** hin, Werte bei **0** auf **normale Datenpunkte**
  + Wenn alle Punkte bei ca. **0,5** liegen, sind Ausreißer **nicht deutlich** zu erkennen
* Algorithmus:
  + Text

# Vergleich

## Intuitiver Vergleich durch Veranschaulichung am Programm

*(Veranschaulichung am Programm durchführen)*

## Laufzeit

### DBSCAN

* Laufzeit des Kern-Algorithmus: O((C +) n\*Q + Σiri)
  + (C: Erstellungszeit („Construction time“) des Indizes, zum Beispiel einen Umgebungsabfrage unterstützenden R\*-Baum)
  + Q:
    - Komplexität der RangeQuery („Umgebungsabfrage“)
    - Q ∈ Θ(n · D)
    - Für einen **gegebenen Datenpunkt** wird die **Entfernung** zu **jedem anderen Datenpunkt** ermittelt
  + Σiri: Summe der Ergebnisgrößen der RangeQuery

### iForest

* **Lineare Zeitkomplexität** mit einer niedrigen Konstante:
  + Waldaufbau: O(t ψ log ψ) ∈ O(1)
  + Auswertung: O(n t log ψ) ∈ O(n)

## Speicherkomplexität

### DBSCAN

* Speicherkomplexität von O(n (+ I)) (Clusterlabels + Index) (1)

### iForest

* Speichernutzung des Waldes ist linear (n Blattknoten und n-1 innere Knoten): **O(2n - 1) ∈ O(n)**

## ROC & AUC

* “Receiver Operating Characteristic”
* “Area under the Curve”
* AUC ist Kennzahl zur Bewertung der Güte eines Klassifikators
* True-Positive-Rate (TPR) gegen False-Positive-Rate (FPR) auftragen
* TPR ist der Anteil der korrekt als positiv eingestuften Datenpunkten an allen wirklich positiven Datenpunkten
* FPR ist der Anteil der fälschlicherweise als positiv eingestuften Datenpunkte an allen wirklich negativen Datenpunkten
* Eine perfekte Klassifikation zeichnet sich aus durch: AUC = 1 🡺 TPR = 1, FPR = 0
* Eine zufällige Klassifikation strebt bei n 🡪 ∞ gegen TPR = FPR für alle FPR ∈ [0, 1] 🡺 AUC = 0.5

## Weiteres

### DBSCAN

* Vorteile
  + Findet nebenbei Cluster
  + Die Anzahl der Cluster muss nicht manuell festgelegt werden, sondern wird automatisch ermittelt. (wiki)
* Nachteile:
  + Die Parameter epsilon und minPts müssen gut gewählt werden, um eine gute Erkennung von Ausreißern zu erzielen, d.h. möglichst keine Falsch-Positiven und möglichst alle Wahrhaft-Positiven. Dazu gibt es eine Heuristik, die aber menschliche Intuition einbezieht, was man daher nicht so einfach automatisieren kann.
  + DBSCAN verwendet **global** einen **festen Dichte-Schwellenwert** und passt sich somit nicht Clustern **unterschiedlicher Dichte** an. Es können z.B. Ausreißer um ein sehr dichtes Cluster herum die gleiche Dichte haben, wie normale Datenpunkte eines abgelegenen anderen Clusters.

### iForest

* Vorteile:
  + Sind **hocheffektiv** und **hocheffizient** (nur **wenige iTrees** und nur eine **kleine Sub-Sampling-Größe** werden dafür benötigt). (6)
  + Sind auch anwendbar bei **extrem großen Datenmengen** und Daten mit **vielen Dimensionen**

# Fazit

Isolation Forests sind mit linearer Laufzeit hocheffektiv und hocheffizient. Ihre Fähigkeit, genau die Ausreißer zu identifizieren, ist mit hohen AUC-Werten sehr gut. Außerdem ist die Parameterbestimmung im Gegensatz zu DBSCAN sehr einfach. IForests ist die richtige Wahl für große Datenmengen, mit vielen Attributen und Daten, über deren Verteilung und Ausreißeranteil man wenig weiß.

DBSCAN identifiziert Ausreißer und teilt nebenbei alle normalen Datenpunkte automatisch in Cluster ein. Dabei spielt es keine Rolle, welche Form ein Cluster hat. Der Rechenaufwand ist quadratisch und der Algorithmus somit nicht für große Datenmengen nutzbar. Außerdem muss die Parameter angeben, die nicht trivial zu ermitteln sind.

# Quellen

1. https://docslib.org/doc/11822058/19-dbscan-revisited-revisited-why-and-how-you-should-still
2. file:///C:/Users/Jonas%20Brauer/Downloads/v91i01.pdf
3. https://eprints.whiterose.ac.uk/767/1/
4. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8786096
5. https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf?q=isolation-forest
6. https://www2.cs.uh.edu/~ceick/7363/Papers/dbscan.pdf
7. https://scikit-learn.org/stable/auto\_examples/miscellaneous/plot\_outlier\_detection\_bench.html#sphx-glr-auto-examples-miscellaneous-plot-outlier-detection-bench-py