Masterarbeit

Erkennung von Fahrspuren mittels Fahrzeugtrajektorien aus Luftaufnahmen

im Master-Studiengang Informatik
der Hochschule Furtwangen

Steffen Schmid

Zeitraum: Wintersemester 2018
Prüfer: Prof. Dr. Christoph Reich
Zweitprüfer: Stefan Kaufmann

Firma: IT-Designers GmbH
Betreuer: Stefan Kaufmann

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 22. Oktober 2018 _____

Unterschrift

Zitat

„Some fancy quote“

Foobar Muman

Danksagung

Kurzfassung

Schlagworte:

Inhaltsverzeichnis

Kurzfassung v

1 Einleitung 1
1.1 Rahmen der Arbeit . . . 1
1.1.1 Das Projekt MEC-View . . . 1
1.1.2 MEC-View Luftbeobachtung . . . 1
1.2 Motivation und Ziele . . . 1
1.3 Aufbau dieser Arbeit . . . 1

2 Grundlagen 2
2.1 Verkehrsanalyse mittels Luftaufnahmen . . . 2
2.2 Rekonstruktion von Fahrzeugtrajektorien aus Luftaufnahmen . . . 2
2.3 Datenaufbereitung und Bereinigung . . . 4
2.4 Clusteranalyse . . . 4
2.4.1 Eigenschaften von Cluster-Sets und Clustern . . . 6
2.4.2 Cluster-Algorithmen . . . 7
2.4.3 Distanzmaße zum Vergleich von Fahrzeugtrajektorien . . . 14
2.5 Untersuchung möglicher Straßentopologien . . . 14

3 Stand der Technik 15
3.1 Clustering von Trajektoriedaten . . . 15
3.2 Erkennung von Fahrspuren . . . 15
3.3 Klassifizierung von Fahrspuren . . . 15
3.4 Defizite vorhandener Lösungen und benötigte Neuerungen . . . 15

4 Clustering von Fahrzeugtrajektorien 16
4.1 Vorverarbeitung der Roh-Trajektorien . . . 16
4.2 Gruppierung der Trajektorien . . . 16

5 Fahrbahn-Bestimmung aus Trajektorie-Clustern 17

6 Fahrbahn Klassifizierung 18

7 Realisierung LaneDetection in MEC-View TrackerApplication Software 19

8 Ergebnisse und Auswertung 20

9 Zusammenfassung und Fazit 21

Literaturverzeichnis 22

Abbildungsverzeichnis

2.1 Übersicht Tracking mit Klassifikator 3
2.2 a) Haar-ähnliche Merkmale b) Beispiele für erkannte Regionen in einem Gesicht 4
2.3 Rohdaten (links) und erwünschtes Clustering-Ergebnis (rechts) 5
2.4 Ablauf einer Clusteranalyse 5
2.5 Agglomeratives Clustering dargestellt als Dendrogramm und geschachteltes Cluster-Diagram 8
2.6 Funktionsweise von k-Means 10
2.7 Darstellung des EM-Cluster-Algorithmus über mehrere Iterationen 12
2.8 Schritte des DBSCAN Algorithmus 13
2.9 Erreichbarkeit in DBSCAN 13

Listings

1 Einleitung

Staus und zäh fließender Verkehr sind sowohl auf Schnell- und Autobahnen, als auch in Städten ein großes Problem und Ärgerniss für Autofahrer. Sie kosten diese nicht nur wertvolle Zeit, sondern auch viel Geld. Laut einer Studie von [Cookson u. a.] kostet Stau jeden deutschen Autofahren pro Jahr durchschnittlich 1770 €. In Summe ergeben sich hieraus beinahe 80 Milliarden Euro an Kosten. Stau ist allerdings nicht nur finanziell für Privatpersonen oder auch Unternehmen ein großer Faktor, sondern er erhöht auch das Unfallrisiko und trägt maßgeblich zur schlechten Luftqualität in Innenstädten bei. Aufgrund längerer Fahrzeiten und der häufigen Be- und Entschleunigung, steigt der Kraftstoffverbrauch der Fahrzeuge und dadurch auch die Schadstoffbelastung in der Luft [Hemmerle, 2016].

Um Stau so gut wie möglich vermeiden zu können, muss man den Verkehr verstehen. Nötig ist ein Verständnis des Straßenverkehrs als Ganzes, sowie der Auswirkungen, welche einzelne Verkehrsteilnehmer und deren Verhalten, auf diesen haben. Hierzu ist das Erstellen von Simulationen sowie die Auswertung realer Verkehrsaufkommen unerlässlich. Die auf diese Weise gesammelten Erkenntnisse bilden die Grundlage, um Straßenabschnitte, insbesondere auch in Innenstädten, intelligent zu gestalten. Des Weiteren können sie eingesetzt werden, um beispielsweise Ampelschaltungen in Städten zu optimieren, wovon auch bestehende Infrastrukturen profitieren können.

Diese Arbeit beschäftigt sich mit der Realisierung einer automatischen Fahrspurerkennung aus Luftaufnahmen, welche bei der Analyse von Spurwechselvorgängen zum Einsatz kommt. Hierzu werden die Trajektoriedaten von Fahrzeugen ausgewertet.

1.1 Rahmen der Arbeit

1.1.1 Das Projekt MEC-View

1.1.2 MEC-View Luftbeobachtung

1.2 Motivation und Ziele

1.3 Aufbau dieser Arbeit

2 Grundlagen

In diesem Kapitel werden die für das Verständnis und die Durchführung der Arbeit benötigten Grundlagenthemen vorgestellt. Nach einer kurzen Erläuterung der Möglichkeiten der Verkehrsanalysen mittels Luftaufnahmen, wird daher darauf eingegangen, auf welche Weise die in dieser Arbeit verwendeten Fahrzeugtrajektorien ermittelt werden. Anschließend werden Methoden vorgestellt, welche zur Bereinigung der gewonnenen Daten verwendet werden können. Als wichtiges Mittel zur Identifizierung von Fahrspuren aus Trajektorien werden zudem verschiedene Cluster-Algorithmen und Distanzmaße vorgestellt.

2.1 Verkehrsanalyse mittels Luftaufnahmen

2.2 Rekonstruktion von Fahrzeugtrajektorien aus Luftaufnahmen

Die in dieser Arbeit verwendeten Fahrzeugtrajektorien stammen aus der Anwendung "Tracker-Application" des MEC-View Teilprojektes *Luftbeobachtung*. Nachfolgend wird beschrieben, wie diese aus den Videoaufnahmen rekonstruiert werden.

Die Verfolgung von bewegten Objekten beziehungsweise Fahrzeugen, wird in der "Tracker-Application" mittels *Supervised Tracking* umgesetzt. Bei diesem Verfahren wird ein initial manuell ausgewählter Bildbereich automatisch mit Hilfe eines erlernten Klassifikators verfolgt. Der Klassifikator muss hierbei zwischen Fahrzeugen und der Umgebung unterscheiden können. Das grundlegende Vorgehen dieses Tracking-Ansatzes ist in Abbildung 2.1 dargestellt und kann wie folgt beschrieben werden:

a) Verfolgtes Objekt befindet sich zum Zeitpunkt t an bekannter Position p_1

b) Zum Zeitpunkt $t + 1$: Anwendung des Klassifikators auf Positionen um p_1

c) Erstellen einer *Confidence Map*, welche die Wahrscheinlichkeit darstellt, das verfolgte Objekt gefunden zu haben

d) Updaten des Trackers auf Position des Maxima der *Confidence Map*

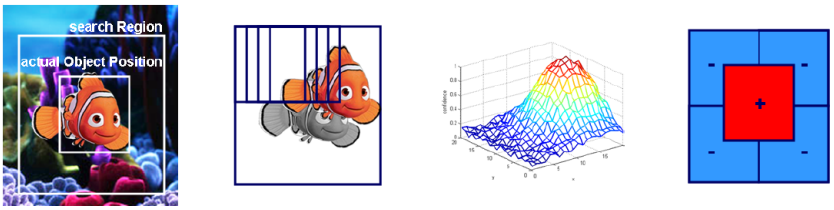


Abb. 2.1: Übersicht Tracking mit Klassifikator [Grabner u. a.]

Das Erlernen eines stabilen Klassifikators in der "Tracker-Application" basiert auf der Arbeit *"Real-Time Tracking via On-line Boosting"* von [Grabner u. a.]. Die Autoren verwenden einen On-line AdaBoost Algorithmus, welcher mehrere *schwache* Klassifikatoren zu einem *starken* Klassifikator kombiniert. Schwache Klassifikatoren müssen hierbei nur eine Erkennungsrate von mehr als 50% besitzen und somit wenig besser als zufallsbedingtes Auswählen sein. Starke Klassifikatoren entstehen durch die Kombination von mehreren schwachen Klassifikatoren. Die Auswahl von schwachen Klassifikatoren erfolgt über sogenante Selektoren, welche aus einer Menge immer jenen wählen, welcher die geringste Fehlerrate bei der Erkennung der Trainings-Objekte besitzen. Der Klassifikator mit der schlechtesten Erkennungsrate wird in jeder Trainingsiteration ersetzt, um das Training zu verbessern. Großer Vorteil der On-line AdaBoost Methode ist, dass sie es ermöglicht, starke Klassifikatoren während des eigentlichen Trackingvorganges zu erlernen. Nach jedem Trackingschritt wird das erfolgreich erkannte Objekt in Trainingssätze zerlegt, auf welche die Klassifikatoren angewandt werden um ihre Performance zu evaluieren. So wird in jedem Schritt die Menge der schwachen Klassifikatoren und der Selektoren aktualisiert. Die Wahl von effizient berechenbaren schwachen Klassifikatoren macht dies möglich.

Die in [Grabner u. a.] und der "Tracker-Application" verwendeten Klassifikatoren sind binär, das heißt, sie teilen Objekte in die zwei Klassen *erkannt* und *nicht erkannt* auf. Konkret werden Haar-ähnliche Bildmerkmale nach [Viola u. a.] als schwache Klassifikatoren verwendet. Diese sind ein Mittel zur Identifikation von Kontrastunterschieden in Bildern, welche sich sehr gut zur Erkennung von Kanten und Linien eigenen. Ein Beispiel der Haar-ähnlichen Merkmale und ihres Einsatzes bei der Gesichtserkennung ist in Abbildung 2.2 dargestellt.

Diese Merkmale werden als schwache Klassifikatoren mit zufälliger Skalierung, Größe und Position auf dem Bild platziert. Sie suchen in dieser Region anschließend nach den von dem Muster definierten Konturunterschieden. Eine Bereich gilt als erkannt, wenn der Betrag der Differenz der Pixelsumme des weißen und schwarzen Bereiches des Musters unter einem festgelegten Grenzwert liegt.

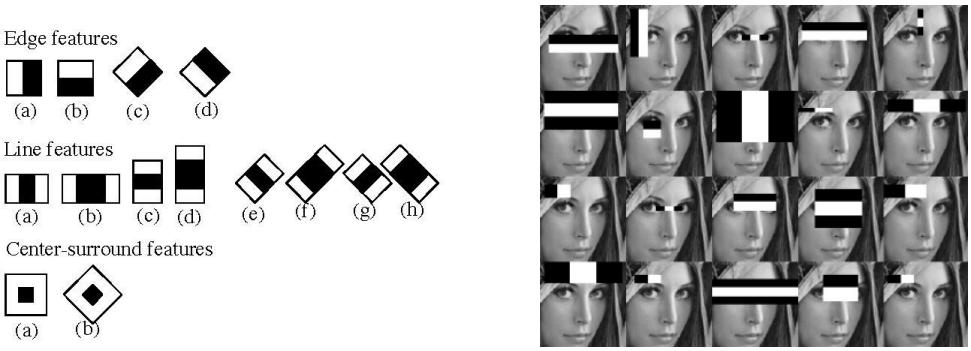


Abb. 2.2: a) Haar-ähnliche Merkmale b) Beispiele für erkannte Regionen in einem Gesicht [Divyansh Dwivedi, 2018]

2.3 Datenaufbereitung und Bereinigung

2.4 Clusteranalyse

Die Clusteranalyse (kurz Clustering) ist ein wichtiges Werkzeug zur Auswertung von Daten unterschiedlichster Art. Sie stellt dabei kein konkretes Vorgehen oder einen Algorithmus dar, sondern beschreibt ein allgemeines Problem, welches auf unterschiedlichste Weise gelöst werden kann. Grundsätzlich ist das Ziel der Clusteranalyse, Datenobjekte aufgrund ihrer Eigenschaften und Beziehungen untereinander so zu gruppieren, dass sich die Objekte einer Gruppe möglichst stark ähneln und sich von den Objekten anderer Gruppen möglichst stark unterscheiden. Je höher die *Homogenität* in einem Cluster und die *Differenz* zwischen den Clustern, desto besser ist die gewählte Clustering Methode. Der Einsatz von Clustering ist in vielen Anwendungsgebieten und in den unterschiedlichsten wissenschaftlichen Disziplinen sehr beliebt, um ein Verständnis für Daten zu erhalten beziehungsweise diese anschließend weiter verarbeiten zu können. So kommt die Clusteranalyse unter anderem in den Feldern des maschinellen Lernens, der Mustererkennung, Bildanalyse, der Biologie (Taxonomie) oder im Bereich Data Mining zum Einsatz. [Tan u. a., 2007]

Die Clusteranalyse hat viel mit dem Problem der Klassifizierung von Daten gemein, insofern sie Datenobjekten Label zuordnet. Im Gegensatz zu *überwachten* Klassifizierungsansätzen, wie dem heute populären überwachten Lernen, leiten Cluster-Algorithmen die Label allerdings alleine aus den vorhandenen Daten ab. Es kommen keine Vergleichsobjekte mit bekannten, händisch vergebenen Labeln zum Einsatz. Aus diesem Grund wird die Clusteranalyse auch häufig als *unüberwachte Klassifizierung* bezeichnet. [Tan u. a., 2007]

Das Konzept eines *Clusters* ist nicht genau definiert, was in einer Vielzahl an unterschiedlichen Ansichten und Algorithmen resultiert, welche sich jeweils für andere Anwendungsfälle eignen und verschiedene Eigenschaften besitzen. Hieraus ergibt sich auch die Tatsa-

che, dass Clustering keine selbsttätiger Prozess ist, welcher sich auf einheitliche Weise auf unterschiedliche Probleme anwenden lässt. Jedes Problem erfordert die individuelle und sorgfältige Auswahl eines passenden Algorithmus, eines Distanzmaßes und der richtigen Parameter. Die Bestimmung dieser geschieht iterativ und nicht selten nach dem Prinzip des *Trial and Error*. In Abbildung 2.3 ist beispielhaft ein Datensatz (links) mit – für den Menschen intuitiv ersichtlich – 7 unterschiedlichen Clustern (rechts) dargestellt. Nach [Jain, 2010] kann allerdings kein verfügbarer Clustering Algorithmus diese alle erkennen. [Jain u. a., 1999; Tan u. a., 2007]

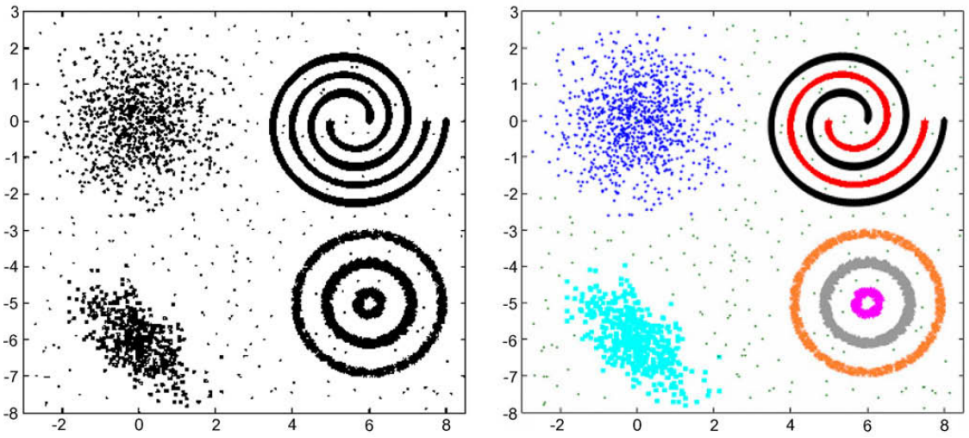


Abb. 2.3: Rohdaten (links) und erwünschtes Clustering-Ergebnis (rechts) [Jain, 2010]

Aufgrund der Limitationen, welche alle Cluster-Algorithmen besitzen, muss der Analyst sich vor deren Anwendung intensiv mit den zu verarbeitenden Daten beschäftigen. Er muss ein Verständnis dafür besitzen, welche Struktur die Daten besitzen, beziehungsweise annehmen können, und nach welchen Mustern zu suchen ist. Besonders wichtiger ist zudem auch die Auswahl der richtigen, das heißt relevanten, Datenmerkmale (*"Feature Selektion"*) und die Wahl deren Repräsentation (*"Feature Transformation"*). Die Selektion und gegebenenfalls Transformation der Daten muss in einem Vorverarbeitungsschritt geschehen, dessen Qualität einen maßgeblichen Einfluss auf das finale Clustering Ergebnis hat. Basierend auf vorangegangener Beschreibung und [Jain u. a., 1999], lässt sich der Ablauf einer Clusteranalyse wie folgt darstellen:

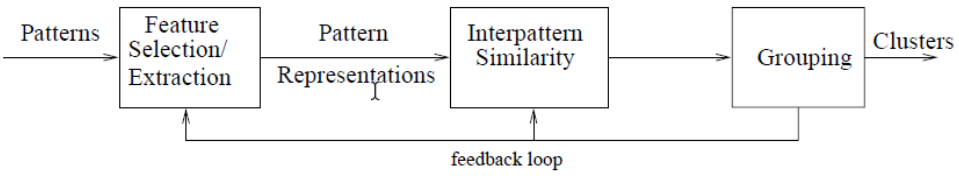


Abb. 2.4: Ablauf einer Clusteranalyse

[Jain, 2010] nennt einige weitere Herausforderungen, welchen man sich bei der Clusteranalyse bewusst sein muss:

- Daten können Ausreißer enthalten. Wie sollen diese behandelt werden?
- Die Anzahl der Zielcluster ist üblicherweise nicht bekannt.
- Validierung der gefundenen Cluster

2.4.1 Eigenschaften von Cluster-Sets und Clustern

Das aus einer Analyse resultierende Cluster-Set und die einzelnen Cluster selbst, können in verschiedene Kategorien unterteilt werden beziehungsweise unterschiedliche Eigenschaften besitzen. Nachfolgend sind die wichtigsten basierend auf [Tan u. a., 2007] und [Jain u. a., 1999; Jain, 2010] aufgeführt.

Cluster-Sets

Bei Cluster-Sets kann grundsätzlich zwischen nachfolgenden Eigenschaften unterschieden werden.

Hierarchisch vs. Partitioniert Von *hierarchischen* Cluster-Sets wird gesprochen, wenn die einzelnen Cluster verschachtelt sind und dabei eine Baum-Struktur bilden. Cluster sind hingegen *partitioniert*, wenn keine Überlagerungen zwischen ihnen existiert.

Exklusiv vs. Überlappend vs. Fuzzy *Exklusive* Cluster-Sets liegen vor, wenn jedem Datenwert ein oder kein Zielcluster zugeordnet wird. Im Gegensatz hierzu können bei *überlappenden* Cluster-Sets Objekte einer oder mehrerer Gruppen angehören. Bei dem sogenannten *Fuzzy* oder *Soft* Cluster-Sets, gehört ein Datenobjekt einem Cluster mit einer bestimmten Wahrscheinlichkeit oder Gewicht an. Algorithmen, welche Daten eine Wahrscheinlichkeit für die Zugehörigkeit zu einem Cluster zuweisen, werden *probabilistische* Cluster-Algorithmen genannt.

Komplett vs. Partielle Von *kompletten* Cluster-Sets wird gesprochen, wenn jedes Element der Eingangsdaten einem Cluster zugeordnet wird. Bei *partiellen* Sets ist dies nicht der Fall. Hier kann ein bestimmter Anteil an Datenwerten als Ausreißer markiert werden, welche keine Gruppe besitzen.

Cluster

Da, wie oben erwähnt, nicht klar definiert ist, was ein Cluster ausmacht, können auch diese unterschiedliche Eigenschaften besitzen. Die wichtigsten Cluster-Arten sind nachfolgend erläutert.

Klar separierte Cluster Unter *klar separiereten* Clustern versteht man solche, in welchen jedes Datenelement einen geringeren Abstand zu allen anderen Elementen des Clusters hat, als zu Elementen außerhalb des Clusters. Diese idealistische Definition eines Clusters ist nur dann erfüllt, wenn die in den Daten enthaltenen Cluster einen großen Abstand voneinander haben. Dies ist in der Realität allerdings selten der Fall.

Prototyp basierte Cluster Von einem *Prototyp basierten* Cluster wird gesprochen, wenn alle Elemente einer Gruppe einen geringeren Abstand zu einem Prototyp oder Referenzwert des Clusters besitzen, als zu denen anderer Gruppierungen. Ein solcher Prototyp ist üblicherweise der Mittelwert der Datenelemente eines Clusters (*Centroid*).

Graphen basierte Cluster Die Definition eines *Graphen basierten* Clusters kann immer dann verwendet werden, wenn Daten als vernetzter Graph dargestellt werden. In einem solchen sind die Elemente Knoten und die Kanten repräsentieren Beziehungen zwischen ihnen. Ein Cluster in einem solchen Graphen ist definiert als Menge von Knoten, welche untereinander verbunden sind, jedoch keine Verbindungen zu Elementen außerhalb des Clusters haben.

Dichte basierte Cluster *Dichte basierte* Cluster sind definiert als Regionen mit einer hohen Dichte an Objekten, welche von Regionen umgeben sind, welche eine geringe Objektdichte besitzen. Elemente, welche in einer solchen Region mit geringen Dichte liegen, welche aus Ausreißer interpretiert. Dichte Bereiche werden üblicherweise gefunden, indem die Nachbarschaften von Elementen untersucht werden.

Konzeptionelle Cluster Eine sehr allgemeine Definition eines Clusters ist die der *konzeptionellen* Gruppen. Hiermit ist gemeint, dass die Elemente eines Clusters einige gemeinsame Eigenschaften besitzen. Dies schließt die oben genannten Cluster-Arten mit ein, lässt sich allerdings beliebig erweitern. So sind beispielsweise in Abbildung XXX x) konzeptionelle Cluster dargestellt, die die Form zweier Kreise und eines Dreiecks haben. Um solche Muster erkennen zu können, würde ein Algorithmus eine besondere Definition eines Clusters benötigen.

2.4.2 Cluster-Algorithmen

Um mit den oben beschriebenen unterschiedlichen Cluster-Set und Cluster Definitionen umgehen zu können, existieren verschiedene Clustering-Modelle. Einige wichtige Clustering-Ansätze sind die Vernetzungs-Modelle, Centroid-basierte-Modelle, Verteilungs-Modelle oder Dichte-Modelle. Für jedes dieser Modelle existieren unterschiedliche Algorithmen. Im Folgenden werden diese Modelle und jeweils exemplarisch ein Algorithmus der diese vertritt vorgestellt.

Vernetzungs-Modelle

Vernetzungs-Modelle werden auch häufig *hierarchische Cluster-Modelle* genannt. Sie beruhen auf der Annahme, dass Elemente, welche nahe beieinander liegen, eine höhere Gemeinsamkeit besitzen als solche, welche weiter voneinander entfernt sind. Zur Bestimmung der Nähe zwischen Elementen benötigen Vernetzungs-Modelle, wie auch andere Cluster-Modelle, eine Definition von Distanz. Diese legt ein sogenanntes *Distanzmaß* fest. Zusätzlich ist ein *Link-Kriterium* notwendig, welches bestimmt, wie genau die Entfernung zwischen zwei Clustern ermittelt wird. Übliche Link-Kriterien sind *Minimum-Linkage*, welches die minimale Distanz zwischen den Objekten der Cluster als Distanz verwendet, oder *Maximum-Linkage* beziehungsweise *Average-Linkage*. [Jain u. a., 1999; George Seif, 2018]

Grundsätzlich teilen sich hierarchische Cluster-Algorithmen in zwei Gruppen auf: *Agglomerative* (Bottom-Up) und *Divisive* (Top-Down) Algorithmen. Agglomerative Ansätze weisen zu Beginn des Cluster-Vorgangs jedem Datenelement eine eigene Gruppe zu und vereinigen diese anschließend. Bei divisiven Ansätzen werden hingegen zu Beginn alle Elemente in einem Cluster zusammengefasst und diese in den nachfolgenden Schritten geteilt.

Als Beispiel wird anschließend der **agglomerative-hierarchische Cluster-Algorithmus** genauer vorgestellt. Sein Vorgehen lässt sich sehr gut anhand sogenannter Dendrogramme oder geschachtelter Cluster-Diagramme darstellen (siehe Abbildung 2.5)

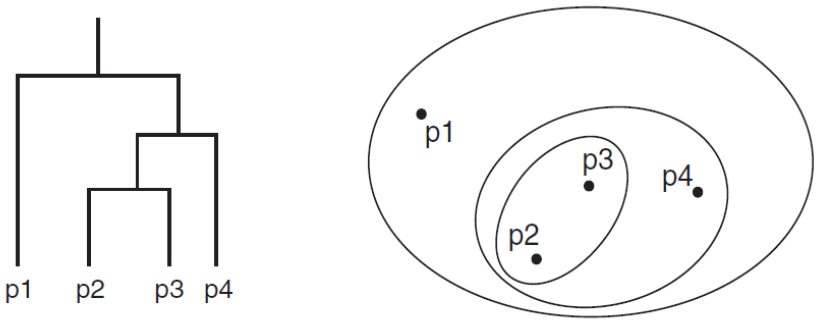


Abb. 2.5: Agglomeratives Clustering dargestellt als Dendrogramm und geschachteltes Cluster-Diagram

Im ersten Schritt des Algorithmus werden alle Datenpunkte als separate Cluster markiert. Diesen Schritt repräsentieren die Blätter des Dendrogramms. Anschließend muss ein Distanzmaß und ein Link-Kriterium gewählt werden. Das am häufigsten verwendete Distanzmaß ist sicherlich der euklidsche Abstand, welcher die Distanz zwischen zwei Punkten oder Vektoren im n-dimensionalen Raum bestimmt. Er ist definiert durch die Formel

2.1.

$$dist(p,q) = \|q - p\|_2 = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2.1}$$

Wird als Link-Kriterium beispielsweise *Minimum-Linkage* gewählt, ist dieses definiert als:

$$link(P, Q) = min\{dist(p,q) : p \in P, q \in Q\} \tag{2.2}$$

Hierbei entsprechen P und Q zwei Clustern, welche die Elemente $p \in P$ und $q \in Q$ enthalten. Auf Basis des gewählten Link-Kriteriums kann nun eine Distanz-Matrix für die einzelnen Cluster erstellt werden. Die zwei Cluster mit minimalem Abstand voneinander werden anschließend zusammengeführt und die vorherigen Schritte werden wiederholt, bis nur noch ein Cluster (Wurzel des Dendrogramms) beziehungsweise die gewünschte Clusteranzahl übrig ist. [George Seif, 2018; Tan u. a., 2007]

Bei den meisten Varianten des agglomerativen Clusterings muss der Nutzer die Anzahl der Zielcluster im vorraus festlegen, was problematisch ist, da diese meist nicht bekannt ist. Umgangen werden kann dies nur, indem ein Link-Kriterium gewählt wird, das ab einer bestimmten Distanz zwischen den Clustern diese nichtmehr fusioniert [George Seif, 2018].

Die Zeitkomplexität des agglomerativen Clusterings beträgt bestenfalls $O(m^2 log\ m)$, weshalb die Menge der Daten, welche mit ihm verarbeitet werden können erheblich begrenzt ist [Tan u. a., 2007].

Centroid-Modelle

Centroid basierte Cluster-Modelle betrachten im Gegensatz zu hierarchischen Modellen nicht die Distanz zwischen Clustern, sondern die Entfernung von Objekten zu Referenzpunkten, sogenannten *Centroids*.

Ein Beispiel für einen Centroid-Cluster-Algorithmus ist **k-Means**. Dieser ist aufgrund seines Alters, seiner Einfachheit und der vielen Weiterentwicklungen wohl der bekannteste Cluster-Algorithmus überhaupt.

Das Ziel von k-Mean ist es, für eine n-dimensionale Punktmenge $X = \{x_1 ... x_n\}$ ein Cluster-Set $C = \{c_1 ... c_k\}$ zu finden, welches die Summe der quadratischen Abweichung (Gleichung 2.3) zwischen allein Punkten in einem Cluster und deren Mittelwert μ_k (Centroids) minimiert.

$$J(c_k) = \sum_{k=1}^{K} \sum_{x_i \in c_k} ||x_i - \mu_k||^2 \tag{2.3}$$

Eine Lösung für dieses Problem zu finden, ist NP-Schwer. Aus diesem Grund ist k-Means ein approximativer Ansatz, welcher nicht garantieren kann, ein globales Minimum zu finden. Die Funktionsweise des Algorithmus ist in Abbildung 2.6 dargestellt. Die Kreuze entsprechen hierbei den Centroids.

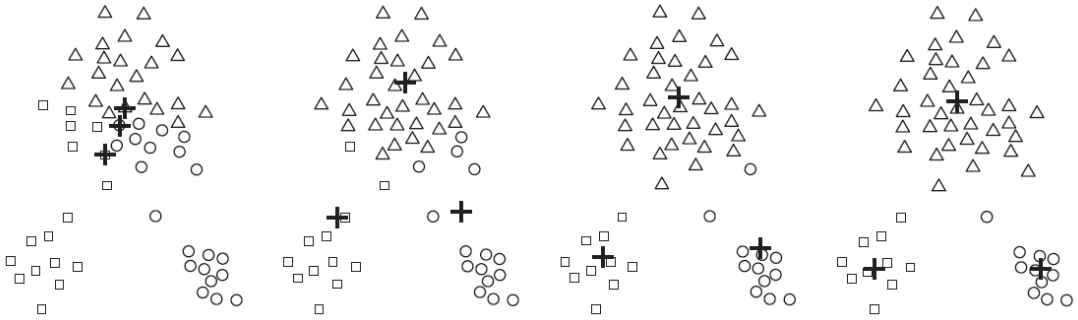


Abb. 2.6: Funktionsweise von k-Means [Tan u. a., 2007]

Ausgehend von der Punktmenge X und der gesuchten Cluster-Anzahl k, werden im ersten Schritt k zufällig positionierte Centroids μ_k definiert. Anschließend wird für alle Punkte x_i der nächstgelegene Centroid μ_j gesucht.

$$j = arg\ min(dist(x_i, \mu_j)) \tag{2.4}$$

x_i wird daraufhin Mitglied in Cluster C_j. Als Distanzmaß ($dist$) kann hier wieder der euklidsche Abstand (Gleichung 2.1) verwendet werden oder aber auch beliebige andere sinnvolle Metriken. Nachdem alle Punkte x_i einem Cluster zugewiesen wurden, werden die Centroid Positionen neu bestimmt. Hierzu wird der Durchschnitt aller Punkte eines Clusters berechnet:

$$c_j = \frac{1}{n} \sum_{x_j \in C_j} x_j \tag{2.5}$$

Diese zwei Schritte werden mehrfach wiederholt, bis das Ergebnis konvergiert, das heißt die Zuweisungen sich nurnoch geringfügig ändern. [Jain, 2010]

Der primäre Nachteil des k-Means Algorithmus ist, das auch bei ihm die Anzahl der Zielcluster spezifiziert werden muss. Desweiteren ist sein Ergebnis aufgrund der zufälligen Initialisierung der Centroids nicht deterministisch. Vorteil von k-Means ist hingegen, dass seine Zeitkomplexität bei $O(n)$ liegt.

Um die genannten Nachteile, zumindest in Teilen, umgehen zu können, existieren diverse Weiterentwicklungen des k-Mean Algorithmus. So stammen beispielsweise von [Hamerly u. a.] und [Pelleg u. a.] die Algorithmen *g-Means* beziehungsweise *x-Means*, welche die Clusteranzahl k auf Basis mehrerer k-Means Durchläufe und statistischer Kennzahlen bestimmen.

Distributions-Modelle

Distributions-Cluster-Modelle basieren auf der Verwendung von statistischen Wahrscheinlichkeitsverteilungen wie beispielsweise der Gauß-Verteilung. Cluster werden darüber definiert, wie wahrscheinlich es ist, dass Objekte der selben Verteilung angehören. Problematisch ist die Verwendung dieser Cluster-Methodik, da sie anfällig für das Problem des *"Overfitting"* ist, wenn die Komplexität der verwendeten Modelle nicht beschränkt wird. Zudem ist die Annahme, dass vielen realen Datensätzen ein statistisches Verteilungsmodell zugrundeliegt, gefährlich. Ist diese These jedoch berechtigt, haben die Modelle den Vorteil, dass sie neben der Zuweisung von Objekten zu Clustern auch Korrelationen zwischen einzelnen Attributen aufzeigen können. [Anders Drachen, 2014]

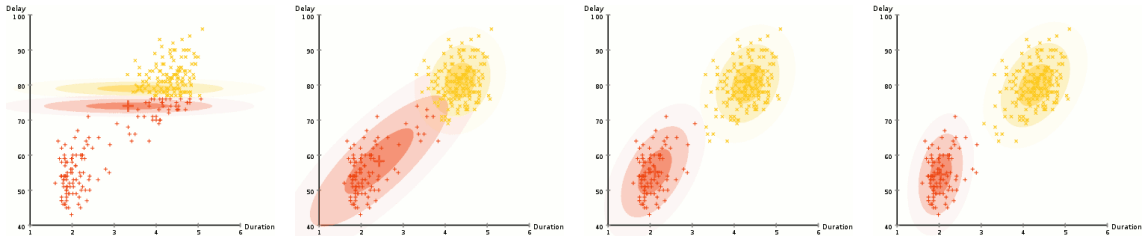
Nachfolgend wird der bekannteste Vertreter der Distributions-Cluster-Algorithmen vorgestellt: das *Expectation–maximization* (EM) Verfahren unter Verwendung sogenannter *Gaussian-Mixture-Models* (GMM). Die Funktionsweise des EM-Algorithmus hat grundsätzlich viel gemein mit der des k-Mean Ansatzes. Es wird ebenfalls mit einer festen Anzahl zufällig initialisierter Modelle gestartet, welche anschließend über mehrere Iterationen an die Daten angepasst werden. Im Gegensatz zu k-Means, sind die gewählten Modelle hingegen Gauß-Verteilungen, welche zwei Parameter besitzen: ihren Mittelwert und die Standardabweichung. Das Vorgehen des EM-Algorithmus ist nachfolgend, basierend auf [George Seif, 2018], beschrieben und in Abbildung 2.7 grafisch dargestellt.

1) Wahl der Clusteranzahl k und Initialisierung der Gauß-Modelle für die entsprechenden Cluster.

2) Berechnung der Wahrscheinlichkeit, dass ein Datenpunkt zu einem Cluster gehört. Je näher ein Datenpunkt dem Zentrum einer Gauß-Verteilung ist, desto höher die Wahrscheinlichkeit für dessen Zugehörigkeit.

3) Basierend auf den Wahrscheinlichkeiten werden die Parameter der Verteilungen neu berechnet. Hierzu wird die gewichtete Summe der Datenpunkt-Positionen errechnet. Die Gewichte entsprechen dabei den Wahrscheinlichkeiten, dass ein Element zu einem Cluster gehört. Hierdurch werden die Gauß-Modelle automatisch den in den Daten enthaltenen Clustern angepasst.

4) Wiederholdung der Schritte 2) und 3), bis das Clustering-Ergebnis konvergiert.



(a) Iteration 1 **(b)** Iteration 2 **(c)** Iteration 3 **(d)** Iteration 4

Abb. 2.7: Darstellung des EM-Cluster-Algorithmus über mehrere Iterationen [George Seif, 2018]

Ziel des EM-Algorithmus ist es, die Parameter der Gauß-Modelle so zu optimieren, dass diese die Verteilung der Daten bestmöglich beschreiben. Am Ende des Clusterings besitzt jeder Datenwert die Zugehörigkeit-Wahrscheinlichkeiten für die einzelnen Cluster. Ein Element wird jenem Cluster zugeordnet, für welches es die höchste Wahrscheinlichkeit besitzt.

Dichte-Modelle

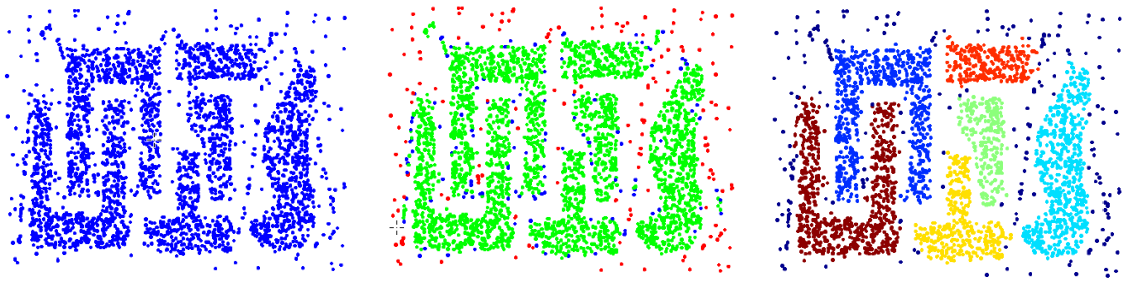
Dichte basierte Cluster sind, wie oben beschrieben, definiert als Regionen hoher Objekt-Dichte, welche von Bereichen geringer Dichte umgeben sind. Dichte-Clustering-Modelle suchen nach eben solchen Regionen. Großer Vorteil der Algorithmen dieser Klasse ist, dass sie Cluster beliebiger Formen finden können, nicht auf die Vorgabe einer Clusteranzahl angewiesen sind und mit Ausreißern umgehen können.

Als Vertreter der Dichte-basierten Ansätze wird nachfolgend der **DBSCAN** Algorithmus (*Density-Based Spatial Clustering of Applications with Noise*), wie in [Gao, 2012] beschrieben, vorgestellt. Er verwendet als Maß für die Dichte einer Region die sogenannte ϵ *-Nachbarschaft* (*Eps*). Diese selektiert für ein Objekt p alle Objekte, welche innerhalb des Radius ϵ um dieses liegen:

$$N_\epsilon(p) = \{q | dist(p,q) \leq \epsilon\} \tag{2.6}$$

Eine ϵ -Nachbarschaft besitzt eine hohe Dichte, wenn in ihr mindestens $MinPts$ Objekte liegen.

Basierend auf der Definition von *Eps*, werden die in einem Datensatz vorhandenen Elemente in drei Klassen unterteilt. Sie sind entweder *Kern-*, *Rand-* oder *Ausreißer-* Objekte. Ein Kernobjekt hat mindestens $MinPts$ andere Punkte in *Eps*. Randobjekte besitzen weniger als $MinPts$ in *Eps*, liegen aber in der Nachbarschaft eines Kernobjektes. Ausreißerobjekte sind weder Kern- noch Randobjekte.



(a) (b) (c)

Abb. 2.8: Schritte des DBSCAN Algorithmus, a) Rohdaten, b) Klassifizierung in Kern- (grün), Rand- (blau) und Ausreißer- (rot) Punkte, c) Cluster Ergebnis [Gao, 2012]

Auf Basis der drei Objektklassen, lässt sich das Prinzip der dichte-basierten *Erreichbarkeit* definieren. Ein Objekt q ist von p *direkt* erreichbar, wenn p ein Kernobjekt ist und q in dessen *Eps* liegt. In Abbildung 2.9 gilt dies beispielsweise für p und p_2. Zwei Elemente sind *indirekt* erreichbar, wenn sie über eine Reihe von Zwischenschritten (direkte Relationen) verbunden sind (transitiv). Dies ist in Abbildung 2.9 für q und p der Fall.

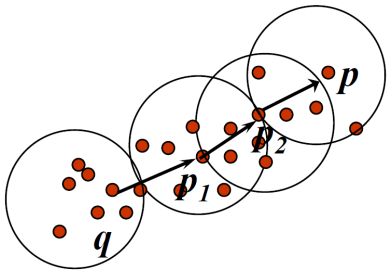


Abb. 2.9: Erreichbarkeit in DBSCAN [Gao, 2012]

Der DBSCAN Algorithmus lässt sich, basierend auf den obigen Definitionen, informell wie folgt beschreiben:

1) Unterteilung der Objekte in die drei Objektklassen. (Abb. 2.8 b))

2) Aussortierung der Ausreißer-Objekte.

3) Wahl eines nicht zugewiesenen Kernobjektes.

4) Erstellung eines neuen Clusters für das Kernobjekt und alle von ihm ausgehend direkt oder indirekt erreichbaren Objekte

5) Wiederholdung der Schritte 3) und 4), bis alle Kern- und Randobjekte einem Cluster zugewiesen sind. (Abb. 2.8 c))

DBSCAN besitzt die oben beschriebenen Vorteile Dichte-basierter Cluster-Algorithmen. Dank einer Zeitkomplexität von $O(nlogn)$ kann er außerdem auch auf große Datensätze angewendet werden. Nachteil des Ansatzes ist hingegen, dass er schlecht mit Clustern umgehen kann, welche unterschiedliche Dichten besitzen.

2.4.3 Distanzmaße zum Vergleich von Fahrzeugtrajektorien

2.5 Untersuchung möglicher Straßentopologien

3 Stand der Technik

3.1 Clustering von Trajektoriedaten

3.2 Erkennung von Fahrspuren

3.3 Klassifizierung von Fahrspuren

3.4 Defizite vorhandener Lösungen und benötigte Neuerungen

4 Clustering von Fahrzeugtrajektorien

4.1 Vorverarbeitung der Roh-Trajektorien

4.2 Gruppierung der Trajektorien

5 Fahrbahn-Bestimmung aus Trajektorie-Clustern

6 Fahrbahn Klassifizierung

7 Realisierung LaneDetection in MEC-View TrackerApplication Software

8 Ergebnisse und Auswertung

9 Zusammenfassung und Fazit

Literaturverzeichnis

[Anders Drachen 2014] Anders Drachen: *Introducing Clustering II: Clustering Algorithms - GameAnalytics*. 2014. – URL https://gameanalytics.com/blog/introducing-clustering-ii-clustering-algorithms.html. – Zugriffsdatum: 2018-10-22

[Cookson u. a.] Cookson, G ; Research, B Pishue I. ; February undefined ; 2017 undefined: Inrix global traffic scorecard. In: *jschultheis.de*. – URL http://jschultheis.de/wp-content/uploads/2018/02/INRIX_2017_Traffic_Scorecard_Report___German.pdf

[Divyansh Dwivedi 2018] Divyansh Dwivedi: *Face Detection For Beginners – Towards Data Science*. 2018. – URL https://towardsdatascience.com/face-detection-for-beginners-e58e8f21aad9. – Zugriffsdatum: 2018-10-18

[Gao 2012] Gao, Jing: Clustering Lecture 4 : Density-based Methods. (2012), S. 1–16. – URL https://cse.buffalo.edu/~jing/cse601/fa12/materials/clustering_density.pdf

[George Seif 2018] George Seif: *The 5 Clustering Algorithms Data Scientists Need to Know*. 2018. – URL https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68. – Zugriffsdatum: 2018-10-19

[Grabner u. a.] Grabner, H ; Grabner, M ; Bmvc, H B. ; 2006 undefined: Real-time tracking via on-line boosting. In: *grabner.family*. – URL http://grabner.family/helmut/papers/Grabner2006RealTimeTracking.pdf

[Hamerly u. a.] Hamerly, G ; Processing, C Elkan A. in neural information ; 2004 undefined: Learning the k in k-means. In: *papers.nips.cc*. – URL http://papers.nips.cc/paper/2526-learning-the-k-in-k-means.pdf

[Hemmerle 2016] Hemmerle, P: Empirische physikalische Eigenschaften des übersättigten innerstädtischen Verkehrs und Energieeffizienz von Fahrzeugen. (2016). – URL https://d-nb.info/1122559259/34

[Jain u. a. 1999] Jain, A. K. ; Murty, M. N. ; Flynn, P. J.: Data clustering: a review. In: *ACM Computing Surveys* 31 (1999), sep, Nr. 3, S. 264–323. – URL http://portal.acm.org/citation.cfm?doid=331499.331504. – ISSN 03600300

[Jain 2010] Jain, Anil K.: Data clustering: 50 years beyond K-means. In: *Pattern Recognition Letters* 31 (2010), jun, Nr. 8, S. 651–666. – URL https://www.sciencedirect.com/science/article/pii/S0167865509002323. – ISSN 0167-8655

[Pelleg u. a.] Pelleg, D ; Icml, AW M. ; 2000 undefined: X-means: Extending k-means with efficient estimation of the number of clusters. In: *cs.uef.fi*. – URL http://cs.uef.fi/{~}zhao/Courses/Clustering2012/Xmeans.pdf

[Tan u. a. 2007] Tan, Pang-Ning u. a.: *Introduction to data mining*. Pearson Education India, 2007

[Viola u. a.] Viola, P ; Recognition, M Jones V. ; Pattern ; CVPR, 2001. ; 2001 undefined: Rapid object detection using a boosted cascade of simple features. In: *ieeexplore.ieee.org*. – URL https://ieeexplore.ieee.org/abstract/document/990517/