AGML, Sommersemester 2020 Vorlesung 20 (6 Seiten)

# Hierarchisches Clustering

Anne Driemel Letzte Aktualisierung: 7. Juli 2020

In der letzten Vorlesung haben wir zwei Beispiele des zentrumsbasierten Clusterings kennengelernt. Beim zentrumsbasierten Clustering messen wir die Ähnlichkeit der Element innerhalb eines Clusters mithilfe eines Zentrums, der diesem Cluster zugewiesen ist. Eine weitere Charakteristik des zentrumsbasierten Clusterings ist die Festlegung auf die Anzahl der Cluster mithilfe eines Parameters $k$. Die Wahl des Parameters hat einen Einfluss auf das berechnete Clustering, was in der Praxis nicht immer gewünscht ist.

Dies wollen wir heute umgehen, indem wir nach hierarchischen Strukturen in der Eingabemenge suchen. Dabei suchen wir nach einer Hierarchie von Partitionierungen (Clusterings) der Eingabemenge, welche ineinander geschachtelt sind. Das heisst, ein Cluster (Teilmenge der Eingabemenge) in einer festen Ebene der Hierarchie sollte in einer höheren Ebene der Hierarchie nicht wieder geteilt werden, sondern dies sollte nur in tieferen Ebenen der Hierarchie passieren.

Beim zentrumsbasierten Clustering wird die Qualität des Clusterings mithilfe einer Zielfunktion beschrieben, welche vom Algorithmus minimiert werden soll. Ähnlich müssen wir nun mathematisch definieren, was die Qualität eines hierarchischen Clusterings ausmacht. Wir definieren dafür drei Eigenschaften, die gegeben sein müssen.

## 1 Definition

Sei $X$ eine Grundmenge und sei $d(\cdot,\cdot)$ eine Metrik auf $X$. Ein hierarchisches Clustering einer $n$-elementigen Menge $S \subseteq X$ ist eine geordnete Menge $\mathcal{C} = \{C_1, \ldots, C_n\}$ mit den folgenden Eigenschaften:

(i) (Jede Menge ist eine Partitionierung von $S$)
Für alle $1 \leq i \leq n$ hat $C_i$ die Form $\{A_1, \ldots, A_{n-i+1}\}$ mit

(a) $\bigcup_{1 \leq j \leq n-i+1} A_j = S$ und
(b) für $j \neq k$ ist $A_j \cap A_k = \emptyset$,

(ii) (Die Mengen sind hierarchisch geschachtelt)
$C_n = \{S\}$ und für alle $1 \leq i < n$ und $A \in C_i$ existiert ein $B \in C_{i+1}$ mit $A \subseteq B$

(iii) (Die Mengen sind optimal)
Für alle $1 \leq i \leq n$ maximiert $C_i$ die folgende Zielfunktion

$$\phi(C_i) = \min_{A,B \in C_i} \min_{a \in A, b \in B} d(a,b)$$

über alle möglichen Partitionierungen von $S$ die aus genau $|C_i|$ Teilmengen von $S$ bestehen.

Die Inklusionsrelationen des hierarchisches Clusterings werden oft als Baum dargestellt. Dafür definieren wir einen Graphen $G = (V, E)$, mit Knotenmenge $V$ und Kantenmenge $E \subseteq V \times V$. Die Knoten des Graphen sind wie folgt definiert. Für jede Teilmenge von $S$ die in einer der Mengen $C_i$ existiert, existiert ein Knoten in $V$. Der Knoten ist mit dieser Teilmenge eindeutig assoziiert. Eine Kante existiert zwischen zwei Knoten $A \in C_i$ und $B \in C_{i+1}$ wenn $A \subset B$. Aus den oben definierten Eigenschaften kann man ableiten, dass der Graph ein Baum ist. Wir legen die Wurzel des Baumes fest als die Menge $S$. Abbildung 1 zeigt ein Beispiel. Beachte dass das Clustering $C_1, \ldots, C_n$ nicht eindeutig dargestellt sind, sondern nur die Inklusionsrelationen zwischen den Mengen in $\bigcup_{1 \leq i \leq n} C_i$.

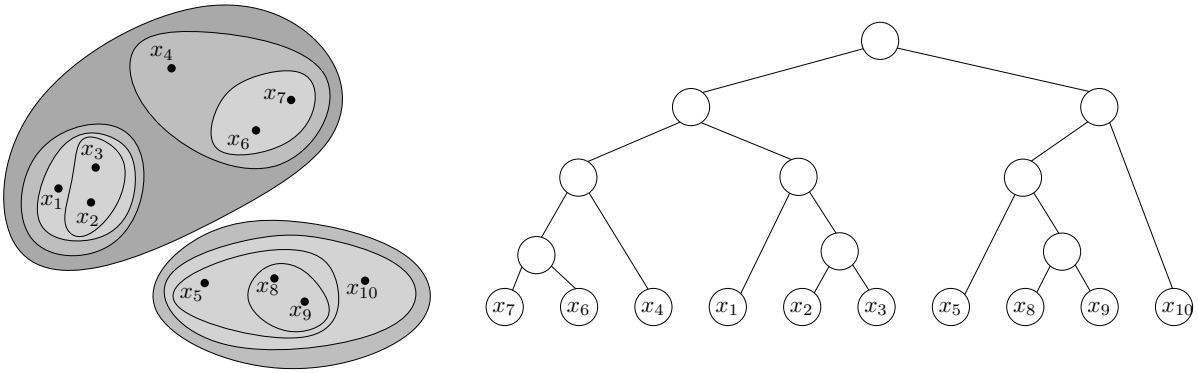


Abbildung 1: Beispiel eines hierarchischen Clusterings einer Menge $\{x_1, \ldots, x_{10}\}$ mit Baumdarstellung der Inklusionsrelationen. Das Clustering $C_1, \ldots, C_n$ lässt sich anhand der Baumdarstellung nicht eindeutig herleiten.

## 2 Algorithmus

Wir betrachten einen Greedy-Algorithmus, der die Partitionierungen $C_1, \ldots C_n$ iterativ berechnet. Er startet mit der Partitionierung in der jedes Element der Eingabemenge eine einzelne Menge darstellt. In jedem Schritt werden die zwei Mengen vereinigt, die den kleinsten Abstand haben, wobei der Abstand zwischen zwei Mengen $A$ und $B$ definiert ist als der kleinste Abstand, zwischen zwei Elementen $a \in A$ und $b \in B$. Wegen dieser Definition des Abstands wird diese Art von Clustering auch *Single-Link-Clustering* genannt. Wir besprechen zunächst die Korrektheit bezüglich der drei Clustering-Eigenschaften. Danach besprechen wir, wie der Algorithmus effizient implementiert werden kann.

`Single-Link-Clustering`$(S = \{x_1, \ldots, x_n\})$
1. Initialisiere $C_1 = \{\{x_1\}, \ldots, \{x_n\}\}$
2. **for** $i$ **in** $2 \ldots n$ **do**
3. Finde zwei Mengen $A, B \in C_i$ welche $\min_{a \in A, b \in B} d(a, b)$ minimieren.
4. Sei $C_{i+1}$ dieselbe Menge wie $C_i$, nur dass $A$ und $B$ vereinigt sind
5. Return $C_1, \ldots, C_n$

**Satz 20.1.** *Die vom Algorithmus berechnete Menge $\mathcal{C} = \{C_1, \ldots, C_n\}$ erfüllt die Clustering-Bedingungen (i),(ii) und (iii).*

*Beweis.* Die Bedingungen (i) und (ii) lassen sich leicht durch Induktion beweisen. Wir konzentrieren uns uns auf den Beweis der dritten Bedingung. Wir behaupten, dass die folgende Schleifen-Invariante gilt:

**Behauptung 20.2.** *Sei $A \in C_i$ und seien $a, b \in A$. Dann existieren Elemente $c_1, \ldots c_\ell \in A$ für ein $\ell \in \mathbb{N}$ mit $a = c_1$, $b = c_\ell$ und $d(c_j, c_{j+1}) \leq \phi(C_i)$ für alle $1 \leq j < \ell$. Wir bezeichnen die geordnete Menge $\{c_1, \ldots, c_\ell\}$ als Pfad.*

Wir stellen den Beweis von Behauptung 20.2 hintan und kommen später darauf zurück.

Sei nun $C'$ eine Partitionierung von $S$ in $|C_i|$ Mengen, welche sich von $C_i$ unterscheidet. Es müssen zwei Punkte $a, b \in S$ existieren, die in $C'$ in unterschiedlichen Mengen liegen, aber in $C_i$ in derselben Menge $A$ liegen. Wenn dies nicht der Fall wäre, dann wären alle Mengen von $C'$ Teilmengen von Mengen in $C_i$. Da beide Partitionierungen dieselbe Anzahl von Mengen enthalten und unterschiedlich sind kann das nicht sein.

Laut Behauptung 20.2 sind $a$ und $b$ in $A$ durch einen Pfad verbunden, der nur kurze Kanten enthält. Wir betrachten den Pfad in $C'$. Es muss entlang des Pfades eine Kante zwischen zwei Punkten $a'$ und $b'$ geben, die in $C'$ nicht in derselben Menge liegen. Es gilt

$$\phi(C') = \min_{A,B \in C'} \min_{a \in A, b \in B} d(a,b) \leq d(a',b') \leq \phi(C_i)$$

Daraus folgt, dass $\phi(C_i)$ optimal ist. Somit ist die dritte Bedingung unter Annahme von Behauptung 20.2 erfüllt. □

# 3 Exkurs: Minimale Spannbäume

Für die effiziente Berechnung der hierarchischen Clusterings betrachten wir zunächst minimale Spannbäume.

**Definition 20.3** (Minimaler Spannbaum)**.** *Sei $G = (V, E)$ ein Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$. Ein Spannbaum von $G$ ist ein zusammenhängender kreisfreier Graph $T = (V, E')$ mit $E' \subseteq E$. Ein minimaler Spannbaum ist ein Spannbaum der die Summe der Kantengewichte $\sum_{e \in E'} w(e)$ minimiert.*

Wir betrachten den Algorithmus von Kruskal zur Berechnung eines minimalen Spannbaumes. Dieser Algorithmus ist seiner Struktur nach dem obigen Algorithmus Single-Link-Clustering sehr ähnlich. Kruskal's Algorithmus berechnet die Kanten des minimalen Spannbaumes iterativ. Er sortiert die Kanten des Graphen aufsteigend nach ihrem Gewicht und bearbeitet sie in dieser Reihenfolge. Für jede Kante testet der Algorithmus ob diese einen Kreis in der bisher gewählten Kantenmenge erzeugen würde. Wenn dem nicht so ist, dann wird die Kante der aktuellen Menge hinzugefügt. Die so berechnete Kantenmenge wird als Ergebnis zurückgegeben.

```
Kruskal-Algorithmus(G = (V, E), w)
1. Sortiere die Kantenmenge E nach ihrem Gewicht
2. Seien e_1, ..., e_m, sodass i ≤ j ⇔ w(e_i) ≤ w(e_j)
3. Sei E' = ∅
4. for i in 1, ..., m do
5.       if e_i erzeugt in (V, E') keinen Kreis
6.             E' = E' ∪ e_i
7. Return (V, E')
```

Um den Test, ob die Kante $e_i$ einen Kreis in der Kantenmenge $E'$ einen Kreis erzeugen würde, effizient ausführen zu können, speichert der Algorithmus zusätzlich die Zusammenhangskomponenten des Graphen $(V, E')$. Sei $e_i$ eine Kante $(a, b) \in E$. Die Kante schliesst in $(V, E')$ genau dann einen Kreis, wenn $a$ und $b$ in derselben Zusammenhangskomponente sind. Für die Zusammenhangskomponenten wird eine Datenstruktur zur Verwaltung von disjunkten Mengen benutzt, welche die folgenden Operationen unterstützt.

- Union$(A, B)$ - Vereinigt die Mengen $A$ und $B$ zu einer neuen Menge und gibt diese zurück.
- Find$(x)$ - Gibt die Menge zurück in der $x$ enthalten ist.

Wir gehen auf diese Datenstruktur nicht weiter ein. Wichtig ist nur, dass Kruskals Algorithmus damit effizient implementiert werden kann. Die Laufzeit des Algorithmus von Kruskal ist in $O(|E| \log |E|)$.

Die folgende grundlegende Eigenschaft von minimalen Spannbäumen hilft dabei, zu zeigen, dass Kruskals Algorithmus einen minimalen Spannbaum berechnet.

**Lemma 20.4** (Schnitt-Eigenschaft)**.** *Sei $G = (V, E)$ ein Graph und sei $U \subseteq E$ eine Teilmenge von Kanten eines minimalen Spannbaums $T = (V, E')$ von $G$. Sei $A \subseteq V$, sodass $U$ keine Kante enthält, die einen Knoten von $A$ mit einem Knoten von $V \setminus A$ verbindet. Sei $F$ die Kantenmenge $\{(a, b) \in E \mid a \in A, b \in V \setminus A\}$. Sei $e = \arg\min_{e \in F} w(e)$. Dann ist $U \cup \{e\}$ Teilmenge eines minimalen Spannbaums von $G$.*

*Beweis.* Angenommen, $e \notin E'$. Da $T$ zusammenhängend ist, muss es einen Pfad in $T$ geben, der die Endpunkte von $e$ verbindet. Dieser Pfad muss auch eine Kante in $F$ haben. Sei $e'$ solch eine Kante. Wir können $e'$ durch $e$ ersetzen und erhalten den Graphen $T' = (V, E' \cup \{e\} \setminus \{e'\})$. Wir können zeigen, dass $T'$ auch ein Spannbaum ist, indem wir zeigen, dass $T'$ zusammenhängend ist und $|V| - 1$ Kanten enthält. Die Summe der Kantengewichte von $T'$ ist

$$\sum_{f \in E'} w(f) - w(e') + w(e) \leq \sum_{f \in E'} w(f)$$

□

**Satz 20.5.** *Kruskals Algorithmus berechnet einen minimalen Spannbaum von $G$.*

*Beweis.* Wir führen eine Induktion über die Kanten aus $E'$ in der Reihenfolge, in der sie vom Algorithmus hinzugefügt werden. Für den Induktionsanfang betrachten wir die erste Kante die hinzugefügt wird. Sei $e = (a, b)$ diese Kante. Wir wenden Lemma 20.4 an mit $U = \emptyset$ und $A = \{a\}$. Daraus folgt, dass $e$ in einem minimalen Spannbaum enthalten ist.

Nun folgt der Induktionsschritt. Sei $e$ eine Kante, die von Kruskals Algorithmus in Zeile 6 zu der Kantenmenge $E'$ hinzugefügt wird. Da $e$ keinen Kreis schliesst, verbindet sie zwei Zusammenhangskomponenten in $E'$, seien diese $A$ und $B$. Da der Algorithmus die Kanten in der aufsteigenden Reihenfolge ihrer Gewichte betrachtet hat $e$ minimales Gewicht unter den Kanten zwischen $A$ und $V \setminus A$. Laut Lemma 20.4 ist $E' \cup \{e\}$ Teil eines minimalen Spannbaums, sofern $E'$ Teilmenge eines minimalen Spannbaums ist. Letzteres folgt aus der Induktionsbehauptung. □

## 4 Anwendung auf das hierarchische Clustering

Sei $G = (V, E)$ ein vollständiger Graph mit Knotenmenge $V = S$. Das Gewicht einer Kante ist der Abstand zwischen den entsprechenden Elementen in $S$, also $w(a, b) = d(a, b)$.[1] Betrachte den Kruskal-Algorithmus angewandt auf diesen Graphen und betrachte den Single-Link-Algorithmus angewandt auf $S$. Bevor wir den folgenden Satz zeigen, müssen wir die Algorithmen noch weiter spezifizieren. Die Anweisung, welche Mengen in Zeile 3 des Single-Link-Clusterings vereinigt werden sollen, ist unter Umständen nicht eindeutig, wenn es Paare von Elementen mit den gleichen Abständen gibt. Gleiches gilt für die Reihenfolge der Kanten in Zeile 2 von Kruskals Algorithmus. Wir nehmen hier einfach an, dass es eine gemeinsame Reihenfolge der Kanten gibt, die von Kruskal verwendet wird und die auch vom Single-Link-Clustering verwendet wird, um zu entscheiden, welcher Abstand am kleinsten ist.

**Satz 20.6.** *Sei $n = |S|$. Seien $e_1, \ldots e_{n-1}$ die Kanten aus $E'$, in der Reihenfolge, in der sie von Kruskals Algorithmus der Menge $E'$ hinzugefügt werden. Für $1 \leq j \leq n$, sei $E_j$ die Kantenmenge $\{e_i \mid 1 \leq i < j\}$. Seien $C_1, \ldots, C_n$ die Mengen, die durch den Single-Link-Clustering Algorithmus berechnet werden. Dann gilt für alle $1 \leq j \leq n$:*

*(i) $C_j$ ist die Menge der Zusammenhangskomponenten des Graphen $(V, E_j)$*

*(ii) Für $j < n$ ist $\phi(C_j) = w(e_j)$*

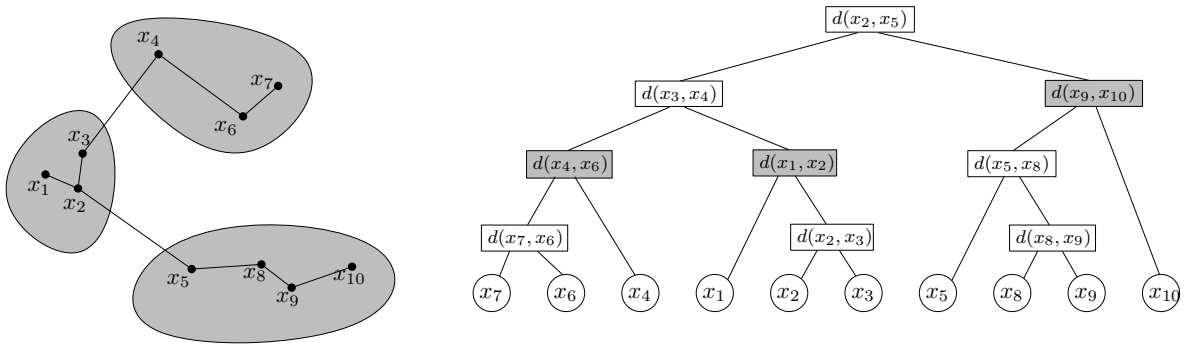


Abbildung 2: Links: Minimaler Spannbaum für die Punktmenge aus Abbildung 1 unter dem Euklidischen Abstand. Rechts: Baumdarstellung der Inklusionsrelationen. Die Abbildung zeigt ausserdem die Menge $C_i = \{\{x_1, x_2, x_3\}, \{x_4, x_6, x_7\}, \{x_5, x_8, x_9, x_{10}\}\}$. In der Baumdarstellung sind die entsprechenden Knoten markiert, deren Unterbäume die Mengen in $C_i$ darstellen. Das Gewicht der Kante $(x_3, x_4)$ bestimmt die Zielfunktion, das heisst $\phi(C_i) = d(x_3, x_4)$.

*Beweis.* Wir führen eine Induktion über $j$. Für $j = 1$ ist die Kantenmenge $E_j$ die leere Menge. Die Menge der Zusammenhangskomponenten ist also die Menge der Knoten des Graphen $G$. Der Single-Link-Clustering Algorithmus definiert die Menge $C_1 = \{\{x_1\}, \ldots, \{x_n\}\}$. Weiter ist $\phi(C_i) = \min_{x_i, x_j \in S} d(x_i, x_j) = w(e_1)$, da die Kante mit kleinstem Gewicht von Kruskal als erstes zu $E'$ hinzugefügt wird. Somit ist der Satz für $j = 1$ erfüllt.

Sei $n > j \geq 1$. Betrachte die Kantenmengen $E_j$ und $E_{j+1}$. Aus der Induktionsannahme wissen wir, dass die Menge $C_j$ die durch den Single-Link-Algorithmus berechnet wird, gleich den Zusammenhangskomponenten des Graphen $(V, E_j)$ ist. Aus der Konstruktion von $E_j$ und $E_{j+1}$ im Satz ergibt sich $E_{j+1} = E_j \cup \{e_j\}$. Wir wollen zeigen, dass $e_j$ genau die zwei Mengen $A$ und $B$ aus $C_j$ verbindet, welche in $C_{j+1}$ zusammenhängend als die vereinigte Menge $A \cup B$ vorkommen.

Betrachte die Vereinigung der Kanten

$$F_j = \bigcup_{A \in C_j} \{(a, b) \in E \mid a \in A, b \in V \setminus A\}$$

Die Kanten in $F_j$ sind genau die Kanten von $G$, die in $(V, E_j)$ keinen Kreis erzeugen. Unter den Kanten in $F_j$ betrachtet der Algorithmus die Kante mit minimalem Gewicht zuerst. Da diese keinen Kreis schliesst, wird sie als nächstes hinzugefügt. Also ist,

$$e_j = \arg\min_{e \in F_j} w(e)$$

Da $C_j$ eine Partitionierung von $V$ ist, folgt

$$w(e_j) = \min_{A \in C_j} \min_{\substack{(a,b) \in E \\ a \in A, b \in V \setminus A}} w((a, b)) = \min_{A, B \in C_j} \min_{\substack{(a,b) \in E \\ a \in A, b \in B}} w((a, b))$$

Da jeder kürzeste Weg zwischen zwei Elementen in verschiedenen Zusammenhangskomponenten über den Schnitt gehen muss, folgt ausserdem

$$w(e_j) = \min_{A, B \in C_j} \min_{a \in A, b \in B} d(a, b)$$

[1] Alternativ kann die Abstandsfunktion auch direkt als die Kürzeste-Wege-Metrik in einem Graphen $G$ mit Knotenmenge $S$ gegeben sein. In diesem Fall betrachten wir $G$ direkt.

Daraus folgt, dass die Kante $e_j$ genau die zwei Mengen in $C_j$ verbindet, die den kleinsten Single-Link-Abstand haben. Also ist $\phi(C_j) = w(e_j)$. □

**Korollar 20.7.** *Kruskals Algorithmus berechnet ein hierarchisches Clustering der Knotenmenge $V$ bezüglich der Abstandsfunktion der kürzesten Wege im Graphen $G$. Die Laufzeit des Algorithmus ist in $O(|E| \log |E|)$. Die Mengen $C_1, \ldots, C_n$ lassen sich mithilfe der Kantengewichte der Kanten des minimalen Spannbaums eindeutig herleiten. Siehe Abbildung 2.*

*Beweis von Behauptung 20.2.* Der Beweis ist nun sehr einfach. Jedes Paar von Knoten in einer Menge von $C_i$ ist durch einen Pfad in der entsprechenden Zusammenhangskomponente von $(V, E_i)$ verbunden. Da Kruskals Algorithmus, die Kanten in der aufsteigenden Reihenfolge ihres Gewichts hinzufügt, ist das Gewicht von jeder Kante entlang des Pfads höchstens $w(e_i)$. Gleichzeitig ist $w(e_i) = \phi(C_i)$, wie wir im obigen Beweis gezeigt haben. □

## Referenzen


- Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani, Algorithms, Kapitel 5.1 (Minimum Spanning Trees)
- Understanding Machine Learning, Kapitel 22.2