Ensemble Clustering

Seminar im Fachgebiet Wissensverarbeitung

Malek Alsalamat

Universität Kassel

1 Was ist Clustering

Clustering ist die Methode im maschinellen Lernen und genauer gesagt "im unüberwachten Lernen", Datenpunkte in Gruppen (Clusters) ohne vorhandenes Wissen zu ordnen. Um das zu schaffen, wird die Ähnlichkeit zwischen die Daten berechnet.

Ziel von Clustering:

- Ähnliche Datenpunkte (Objekte) sollen im selben Cluster sein.
- Datenpunkte im verschiedenen Cluster soll möglichst unähnlich sein.

Also Clustering führt zu der Verringerung von der Komplexität.

Die gefundenen Ähnlichkeitsgruppen können graphentheoretisch, hierarchisch, partitionierend oder optimierend sein. Inzwischen gibt es im Clustering eine große Vielfalt an Methoden, die eingesetzt werden können. Das hängt aber vom Anwendungsfall und der Größe der Datenmenge ab.

Was man aber dabei beachten soll, dass die Algorithmen verschiedene Ergebnisse liefern können [4].

2 K-Means als Beispiel für partitionierende Verfahren

Im partitionierenden Verfahren wird ein Clustering mit k Cluster mit minimalen Kosten gesucht. Und bei K-Means wie der Name bereits sagt, sucht das Algorithmus für jeden Cluster einen zentralen Punkt, bei dem die Varianz zu allen umliegenden Punkten möglichst gering ist. Das Ganze passiert in einem iterativen Verfahren wie folgendes [4]:

- Initialisierung: Zufällige Auswahl von K Zentren.
- Zuweisung aller Datenpunkten zum nächstliegenden Zentrum, gemessen an einer Distanzmetrik. Distanzmetrik berechnet hier die Ähnlichkeit.
- Verschieben der Zentren in den Mittelpunkt aller zugeteilten Datenpunkte.
- Gehe zu 2), außer ein Abbruchkriterium ist erreicht. Der Abbruchkriterium ist, wenn die Zentren sich nicht mehr ändern.

3 Ensemble Clustering

Im Zusammenhang mit maschinellem Lernen ist Ensemble im Allgemein definiert als ein maschinelles Lernsystem, das mit einer Reihe von parallel arbeitenden einzelnen Modellen konstruiert wird. Und die Ergebnisse von diesen Modellen werden mit einer Entscheideungsfusionsstratigie kombiniert, um einzige Lösung für das gegebenen Problem zu liefern.
Die Ensemble Methode war zuerst nur im Bereich des überwachten Lernens benutzt und betrachtet, aber aufgrund seiner erfolgreichen Applikationen in Klassifizierungsaufgaben wurde es versucht, die Ensemble Methode auf unüberwachtes Lernen anzuwenden [1]. Insbesondere auf Clustering Probleme. Die Gründe dafür sind erstens, dass es kein vorhandenes Wissen über die zugrundeliegende Struktur. Zweitens es gibt keinen einzelnen Clustering-Algorithmus, der für verschiedene Probleme konsistent gute Leistungen erbringen kann.
Ensemble Clustering zielt darauf ab, mehrere Clusteringsmodelle zu kombinieren, um ein besseres Ergebnis als die einzelnen Clustering-Algorithmen. Dies ist in vielen Kontexten sehr nützlich [5]:

- Qualität und Robustheit
- Knowledge Reuse
- Distributed Computing

Das Ensemble Clustering Problem lässt sich wie folgt beschreiben:
Gegeben: Mehrere Clusterings aus verschiedenen Algorithmen von derselben Datenmenge.
Gesucht: Ein kombiniertes Clustering, welches so viele Informationen wie möglich mit den gegebenen Clusterings teilt.

4 Das Ensemble Problem

Notation: Sei $X = \{x_1, x_2, \ldots, x_n\}$ eine Menge von Objekten (Datenpunkte). Eine Partitionierung dieser n Objekten in k Clusters kann als eine Menge von k Mengen von Objekten $\{C_l \mid l = 1, \ldots, k\}$ oder als ein Label Vektor $\lambda \in N^n$ dargestellt werden. Ein Clusterer Φ ist eine Funktion, welche einen Label Vektor bei gegebene Objekten liefert.
Eine Mengen von r Clusterings (Labelings) $\lambda^{(1,\ldots,r)}$ ist zu einem Clustering anhand einer Consensus Funktion kombiniert.

4.1 Beispiel

Sei $X = \{x_1, x_2, \ldots, x_7\}$ eine Menge von Objekte und 4 Clusters $\{C_l, C_2, C_3, C_4\}$. Die folgende Label Vektoren spezifizieren vier Clusterings von X.
$\lambda^{(1)} = (C_l, C_l, C_l, C_2, C_2, C_3, C_3)^T$ $\quad$ $\lambda^{(2)} = (C_2, C_2, C_2, C_3, C_3, C_l, C_l)^T$
$\lambda^{(3)} = (C_l, C_l, C_2, C_2, C_3, C_3, C_3)^T$ $\quad$ $\lambda^{(4)} = (C_l, C_2, ?, C_l, C_2, ?, ?)^T$
$\lambda^{(1)}$ wird so gelesen: $x_1, x_2, x_3 \in C_1$ usw.

Man kann erkennen, dass die Label Vektoren $\lambda^{(1)}$ und $\lambda^{(2)}$ logischerweise identisch sind. $\lambda^{(3)}$ liefert einen Unterschied von der Zugehörigkeit der Objekte x_3 und x_5. $\lambda^{(4)}$ weicht sich von der Anderen sehr ab, außerdem $\lambda^{(4)}$ enthält *missing Data* [5].
Gesucht ist jetzt ein kombiniertes Clustering, welches so viele Informationen wie möglich mit den vier gegebenen Clusterings teilt. Intuitiv, ein gutes kombiniertes Clustering in diesem Fall ist $\lambda^{(1)}$ oder auch $\lambda^{(2)}$.

4.2 Objektive Funktion für Ensemble Clustering

Gegeben nun r Gruppierungen mit $\lambda^{(i)}$ ist die $i-te$ Gruppierung, welche k^i Clusters hat. Eine *consensus* Funktion ist wie folgt definiert $\Gamma : \mathbb{N}^{n\times r} \to \mathbb{N}^n$ [5].

$$\Gamma : \{\lambda^i \mid i \in 1, \ldots, r\} \to \lambda. \tag{1}$$

Wenn es grundsätzliche keine Informationen über die relative Bedeutung der einzelnen Gruppierungen gibt, dann ist eine sinnvolle Ziel für die *consensus* Antwort, ein Clustering zu suchen, welches die meisten Informationen mit dem ursprünglichen **Clusterings** teilt.
Im Folgenden brauchen wir zwei Definitionen und zwar von Entropie und gegenseitige Information.
Entropie: ist die minimale Beschreibungskomplexität einer Zufallsvariablen.
Gegenseitige Information: st die Kommunikationsrate im Gegenwart von Rauschen.
Gegenseitige Information, die ein symmetrisches Maß zur Quantifizierung der statistischen Informationen zwischen zwei Distributionen ist, liefert einen Hinweis auf die gemeinsame Informationen zwischen einem Paar von *Clusterings*. Seien X und Y zwei Zufallsvariablen beschrieben durch das Clustering $\lambda^{(a)}$ und $\lambda^{(b)}$ mit $k^{(a)}$ und $k^{(b)}$ Clusters. $I(X,Y)$ beschreibt die gegenseitige Information zwischen X und Y, und $H(X)$ beschreibt die Entropie von X.
Zur Vereinfachung der Interpretation und Vergleichbarkeit ist eine normalisierte Version von $I(X,Y)$, welche zwischen 0 und 1 liegt, gewünscht. Dies folgt zur normalisierten gegenseitigen Information NMI [5]:

$$NMI(X,Y) = \frac{I(X,Y)}{\sqrt{H(X)H(Y)}}. \tag{2}$$

Da $I(X,Y)$ eine Metrik ist, also es gilt $I(X,Y) \leq min(H(X), H(Y))$, ist dann $NMI(X,X) = 1$ genauso wie gewünscht.
Die zweite Gleichung kann durch von *Clusterings* bereitgestellten Strichproben geschätzt werden. Sei $n_h^{(a)}$ die Anzahl von Objekten in Cluster C_h anhand das Clustering $\lambda^{(a)}$ und $n_l^{(b)}$ die Anzahl von Objekten in Cluster C_l anhand das Clustering $\lambda^{(b)}$. Sei $n_{h,l}$ die Anzahl der gemeinsamen Objekten zwischen Cluster

h anhand $\lambda^{(a)}$ und Cluster l anhand $\lambda^{(b)}$. Dann ist [5]:

$$\phi^{(NMI)}(\lambda^{(a)},\lambda^{(b)}) = \frac{\sum_{h=1}^{k^{(a)}}\sum_{l=1}^{k^{(b)}} n_{h,l}\cdot\log\left(\frac{n\cdot n_{h,l}}{n_h^{(a)}\cdot n_l^{(b)}}\right)}{\sqrt{\left(\sum_{h=1}^{k^{(a)}} n_h^{(a)}\log\frac{n_h^{(a)}}{n}\right)\left(\sum_{l=1}^{k^{(b)}} n_l^{(b)}\cdot\log\frac{n_l^{(b)}}{n}\right)}}. \tag{3}$$

Von Gleichung (3) wird einen Maß zwischen einer Menge Λ und einem einzelnen Clustering $\hat{\lambda}$ als die durchschnittliche normalisierte gegenseitige Information definiert (ANMI) [5]:

$$\phi^{(ANMI)}(\Lambda,\hat{\lambda}) = \frac{1}{r}\sum_{q=1}^{r}\phi^{(NMI)}(\hat{\lambda},\lambda^{(q)}). \tag{4}$$

Das optimale kombinierte Clustering $\lambda^{(k-opt)}$ ist das Clustering mit der maximalen durchschnittlichen gegenseitigen Information, wobei k ist die Anzahl von *consensus* Clusters. In anderen Worten, $\phi^{(ANMI)}$ ist unsere objektive Funktion und $\lambda^{(k-opt)}$ ist [5]:

$$\lambda^{(k-opt)} = \arg\max_{\hat{\lambda}}\sum_{q=1}^{r}\phi^{(NMI)}(\hat{\lambda},\lambda^{(q)}), \tag{5}$$

wobei $\hat{\lambda}$ durch alle mögliche $k-Partitionierungen$ läuft.
In unserem Beispiel enthält das Clustering $\lambda^{(4)}$ *missingData*. Für solche Fälle kann das objektive *consensus* Clustering aus der Gleichung (5) durch Berechnung eines gewichteten Durchschnitt von der gegenseitigen Information mit allen bekannten Clusters verallgemeinert werden [5].

$$\lambda^{(k-opt)} = \arg\max_{\hat{\lambda}}\sum_{q=1}^{r}|\Upsilon^{(q)}|\phi^{(NMI)}(\hat{\lambda}_{\Upsilon^{(q)}},\hat{\lambda}^{(q)}_{\Upsilon^{(q)}}). \tag{6}$$

Wobei $\Upsilon^{(i)}$ ist eine Menge von Indizes der Objekten mit bekannten Clusters für das $i-te$ Cluster.

5 Effiziente consensus Funktion

In diesem Abschnitt werden einige Verfahren zur Lösung des Ensemble Clustering Problems vorgestellt. Es gibt zwei Arten von *consensus* Funktionen, eine basiert auf die Median Partitionierung Approximation und die zweite basiert auf das gemeinsame Auftreten von Objekten, die sogenannte *objects co−occurrence approach* [6].
Eine Einordnung von der wichtigsten *consensus* Funktionen werden in *Abbildung*1 vorgestellt. Aber wir werden uns hier nur auf die *Graph and hypergraph based methods* fokussieren.

5.1 Graph and hypergraph based methods

Diese Art von Clustering-Ensemble-Methoden transformieren das Kombinationsproblem zu einem Graph bzw. Hypergraph-Partitionierungsproblem. Der Unter-

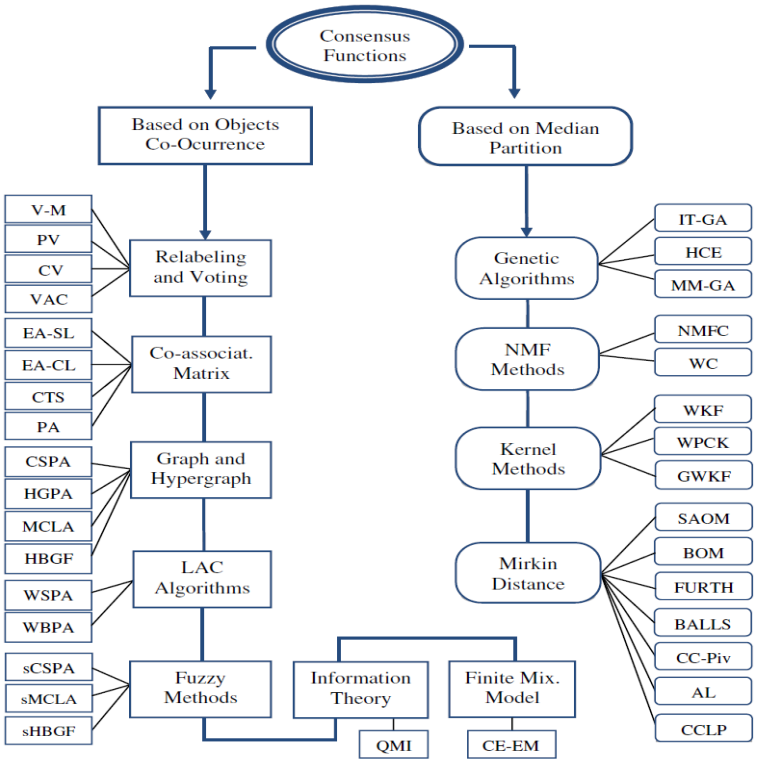

Abb. 1. Schematische Darstellung der wichtigsten Konsensfunktionstechniken. Konsensfunktionen, die auf dem Objekt objects co-occurrence approach basieren, sind durch ein Rechteck (links) dargestellt, die auf der Median Partitionierung Approximation werden durch ein abgerundetes Rechteck dargestellt (rechts) [6].

schied zwischen den Methoden liegt auf die Art und Weise, wie der (Hyper-)Graph aus der Menge der Clusterings aufgebaut wird, und wie Kantenschnitt von der Graph definiert ist, um die *consensus* Partitionierung zu erhalten. Hier werden vier Methoden für diese Art von Clustering vorgestellt und zwar:

- Cluster-based Similarity Partitioning Algorithm (CSPA): Hier wird aus dem Hypergraph eine $n \times n$ Ähnlichkeitsmatrix (die Co-Assoziationsmatrix) konstruiert. Dies kann als eine Adjazenzmatrix eines zusammenhängenden Graphen angesehen werden, wobei die Knoten die Elemente der Datenmenge X sind und die kanten zwischen zwei Objekten ein zugeordnetes Gewicht hat. Das Gewicht entspricht, wie oft sich die Objekte im selben Cluster befinden. Danach wird das Graph-Partitionierungsalgorithmus MEITS verwendet, um die *consensus* Partitionierung zu erhalten [6].
- HyperGraphs Partitioning Algorithm (HGPA): Hier wird den Hypergraphen direkt partitioniert, indem die minimale Anzahl von Hyperkanten elimi-

niert wird. Außerdem wird es davon ausgegangen, dass alle Hyperkanten das gleiche Gewicht haben. Und es wird nach der kleinstmöglichen Anzahl von Hyperkanten gesucht, die den Hypergraphen in k zusammenhängende Komponenten von annähernd gleicher Dimension unterteilen. Für die Implementierung dieser Methode wird die Hypergraphs-Partitionierungspaket $HMETIS$ verwendet [6].

- Meta-CLustering Algorithm (MCLA):Vor allem wird hier die Ähnlichkeit zwischen zwei Clusters C_i und C_j anhand des $Jaccard - Indexes$ bezüglich der Menge der Objekte definiert. Somit ist die Ähnlichkeitsmatrix zwischen Clusters gebildet, welche die Adjazenzmatrix des Graphen darstellt. Aber unter der Berücksichtigung, dass die Clusters als Knoten sind und der Kante zwischen zwei Knoten gewichtet ist, wobei das Gewicht die Ähnlichkeit zwischen Clusters entspricht. Danach wird dieser Graph mit dem METIS partitioniert und das erhaltene Cluster wird als Meta-Cluster bezeichnet. Schließlich, um die eindeutige Partitionierung zu finden, wird die Anzahl jedes Objektes im Meta-Cluster berechnet und jeder Objekt wird zu einem Meta-Cluster geordnet, in dem er öfter aufgetreten ist [6].
- Hybrid Bipartite Graph Formulation (HBGF): In diesem letzten Algorithmus werden die Clusters und Objekten zusammen in derselben Graph modelliert. Bei dieser Methode wird der bipartite-Graph so gebildet, dass es keine Kanten zwischen Knoten gibt, falls sie beide entweder Objekte oder Clusters sind. Also es gibt nur Kanten zwischen zwei Knoten, falls ein Knoten einem Cluster repräsentiert und der Zweite einem Objekt, der zu diesem Cluster gehört, repräsentiert. Die *consensus* Partitionierung wird unter der Verwendung von METIS oder $Spectral - Clustering$ erhalten [6].

5.2 Darstellung von Gruppen von Clusterings als Hypergraph

Wie wir schon im vorherigen Abschnitt gesehen haben, die Clusterings sollten zu einem Hypergraph umgewandelt werden, damit eine der vier Methoden verwendet werden kann. Und das ist genau unser Ziel in diesem Abschnitt.
Ein **Hypergraph** besteht aus Knoten und Hyperkanten. Eine Kante in einem regulären Graph verbindet *genau* zwei Knoten. Eine **Hyperkante** ist eine Verallgemeinerung einer Kante, da sie jede Menge von Knoten verbinden kann.
Für jeden Label Vektor (*Clustering*) $\lambda^{(q)} \in \mathbb{N}^n$ konstruieren wir die binäre Zugehörigkeitsindikatormatrix $H^{(q)} \in \mathbb{N}^{n \times k^{(q)}}$, wobei jeder Cluster als Hyperkante (*Spalte*) dargestellt wird (Siehe Abb.2). Alle Einträge einer Zeile in der binären Zugehörigkeitsindikatormatrix $H^{(q)}$ werden zu 1 addiert, falls die Zeile mit einem Objekt, welches sein Cluster bekannt ist, übereinstimmt. und Zeilen für Objekte mit unbekanntem Cluster sind mit Null ausgefüllt.
Die Blockmatrix $H = (H^{(1)} \dots H^{(r)})$ definiert die Adjazenzmatrix eines Hypergraphen mit n Knoten und $\sum_{q=1}^{r} k^{(q)}$ Hyperkanten. Jeder Spaltenvektor $\boldsymbol{h_a}$ spezifiziert eine Hyperkante h_a, wobei 1 anzeigt, dass der Knoten mit der entsprechenden Zeile zu der Hyperkante gehört und 0 gibt an, dass dies nicht der Fall ist. Somit haben wir jeder Cluster zu einem Hyperkante und die Menge von

	$\lambda^{(1)}$	$\lambda^{(2)}$	$\lambda^{(3)}$	$\lambda^{(4)}$			$\mathbf{H}^{(1)}$			$\mathbf{H}^{(2)}$			$\mathbf{H}^{(3)}$			$\mathbf{H}^{(4)}$	
							$\mathbf{h}_1$	$\mathbf{h}_2$	$\mathbf{h}_3$	$\mathbf{h}_4$	$\mathbf{h}_5$	$\mathbf{h}_6$	$\mathbf{h}_7$	$\mathbf{h}_8$	$\mathbf{h}_9$	$\mathbf{h}_{10}$	$\mathbf{h}_{11}$
x_1	1	2	1	1		v_1	1	0	0	0	1	0	1	0	0	1	0
x_2	1	2	1	2		v_2	1	0	0	0	1	0	1	0	0	0	1
x_3	1	2	2	?	$\Leftrightarrow$	v_3	1	0	0	0	1	0	0	1	0	0	0
x_4	2	3	2	1		v_4	0	1	0	0	0	1	0	1	0	1	0
x_5	2	3	3	2		v_5	0	1	0	0	0	1	0	0	1	0	1
x_6	3	1	3	?		v_6	0	0	1	1	0	0	0	0	1	0	0
x_7	3	1	3	?		v_7	0	0	1	1	0	0	0	0	1	0	0

Abb. 2. Beispiel für das Cluster-Ensemble-Problem mit $r = 4$, $k^{(1,\ldots,3)} = 3$ und $k^{(4)} = 2$: Original-Label-Vektoren (links) und äquivalente Hypergraphendarstellung mit 11 Hy- peredges (rechts). Jeder Cluster wird in eine Hyperedge umgewandelt [5].

Clusterings zu einem Hypergraph abgebildet [5].

6 Cluster-based Similarity Partitioning Algorithm

In diesem Abschnitt wird das *Cluster−based Similarity Partitioning Algorithm* im Details geklärt. In diesem Algorithmus sollte zuerst eine Ähnlichkeitmatrix konstruiert werden. Basieren auf einer grob aufgelösten Suchtweise haben zwei Objekte eine Ähnlichkeit von 1, falls sie im gleichen Cluster sind und eine Ähnlichkeit von 0 ansonsten. Somit kann man eine $n \times n$ binäre Ähnlichkeitmatrix für jedes Clustering einfach erstellen.
Der eingangsbezogene Durchschnitt von r solchen Matrizen, die die r Gruppen darstellen, ergibt eine Gesamtähnlichkeitsmatrix S mit einer feineren Auflösung. Einträge von S bezeichnen den Anteil der Gruppierungen, in denen sich zwei Objekte im selben Cluster befinden, und können mit einer einzigen dünnbesetzten Matrixmultiplikation ($sparse - matrix$) berechnet werden $\mathrm{S} = \frac{1}{r} HH^T$. Abb.3 veranschaulicht die Erstellung der clusterbasierten Ähnlichkeitsmatrix für das in Abb.2 angegebene Beispiel. Nun können wir die Ähnlichkeitsmatrix verwenden, um die Objekte mit einem beliebigen sinnvollen Clustering-Algorithmus auf der Basis von Ähnlichkeit zu *reclustern*. In unserem Fall entscheiden wir uns für eine Partitionierung des induzierten Ähnlichkeitsgraphen (Knote = Objekt, Kantengewicht = Ähnlichkeit) mit METIS wegen seiner robusten und skalierbaren Eigenschaften.
CSPA ist die einfachste und naheliegendste Heuristik, aber ihr Rechen- und Speicheraufwand Komplexität sind beide quadratisch in $O(n^2)$ [5].

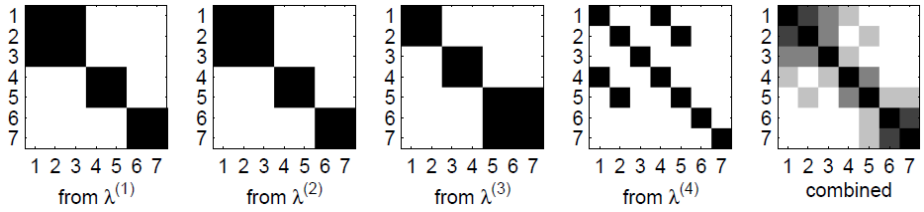

Abb. 3. Veranschaulichung von (CSPA) für das in Abb.2 angegebene Cluster-Ensemble-Beispielproblem. Jedes Clustering besitzt eine Ähnlichkeitsmatrix. Die Matrixeinträge werden durch Dunkelheit proportional zur Ähnlichkeit dargestellt. Ihr Durchschnitt wird dann verwendet, um die Objekte neu zu clustern und einen Konsens zu erhalten [5].

6.1 Mehrstufige Graphenbisektion

Ein mehrstufiger Graphbisektionsalgorithmus funktioniert wie folgt: betrachte einen gewichteten Graphen $G_0 = (V_0, E_0)$ mit Gewichten sowohl aus Knoten als auch auf Kanten. Dieser Algorithmus besteht aus den folgenden frei Phasen [3]:

- Vergröberungsphase *Coarsening*: Der Graph G_0 wird in ein Folge von kleineren Graphen $G_1, G_2, \ldots, G_m$ umgewandelt, so dass $|V_0| > |V_1| > \ldots > |V_m|$.
- Partitionierungsphase: Eine 2-fache-Partition P_m des Graphen $G_m = (V_m, E_m)$ wird berechnet, die V_m in zwei Teile unterteilt, die jeweils die Hälfte der Knoten von G_0 enthält.
- Entgröberungsphase *Uncoarsening*: Die Partition P_m von G_m wird auf G_0 zurückprojiziert, indem man durch die Zwischenpartitionen $P_{m-1}, P_{m-2}, \ldots, P_1, P_0$ geht.

6.1.1 Vergröberungsphase (*Coarsening*) :

In diesem Unterabschnitt beschreiben wir die grundlegende Ideen hinter der Vergröberung mit Hilfe von *Matchings*.
Bei einem Graphen $G_i = (V_i, E_i)$ kann man einen gröberen Graphen durch Kollabieren benachbarter Knoten erhalten. So wird die Kante zwischen zwei Knoten kollabiert und ein $Multi - Knote$ wird aus diesen beiden Knoten erstellt. Diese Idee von Kollabieren von Kanten lässt sich in Form von *Matching* beschrieben werden.
Der nächste gröbere Graph G_{i+1} wird also aus G_i konstruiert, indem ein *Matching* von G_i gefunden wird und die Knoten von diesem *Matching* zu einem $Multi - Knote$ kollabieren. Alle Konten, die nicht im Matching waren, werden einfach im G_{i+1} kopiert. Da das Ziel des Kollabieren von Knoten mit Hilfe von *Matching* darin besteht, die Größe des Graphen G_i zu verringern , sollte das *Matching* eine große Anzahl von Kanten enthalten. Das maximale Matching, das die maximale Anzahl von Kanten hat, wird als *maximum Matching* genannt. Da jedoch die Komplexität der Berechnung eines *maximum Matching* im Allgemeinen höher

ist als die Berechnung eines maximalen Matchings ist, wird maximale Matching bevorzugt. Im verbleibenden Unterabschnitt beschreiben wir zwei Methoden, die wir zur Auswahl maximaler Matchings für die Vergröberung.

- Random matching (RM): Ein maximales Matching kann mit einen randomisierten Algorithmus generiert werden. Die Knoten sind dann in zufälliger Reihenfolge besucht. Wenn ein Knoten u noch nicht ausgewählt wurde, dann wählen wir einen seiner nicht ausgewählter Nachbarknoten aus. Wenn es solcher Knoten v existiert, wird die Kante (u, v) zum Matching gefügt und markieren wir die Knoten u und v als besucht. Wenn es keinen solchen Konten v gibt, dann bleibt u wie es ist in der RM. Die Komplexität von diesem Algorithmus liegt in $O(|E|)$.
- Heavy edge matching (HEM): Unser übergeordnetes Ziel ist es jedoch, eine Partition zu finden, die den Kantenschnitt $edge - cut$ minimiert. Betrachten wir einen Graphen $G_i = (V_i, E_i)$, ein Matching M_i, das zur Vergröberung von G_i verwendet wird, und seinen gröberen Graphen $G_{i+1} = (V_{i+1}, E_{i+1})$, induziert durch M_i. Wenn A eine Menge von Kanten ist, so ist $W(A)$ die Summe der Gewichte der Kanten in A. Es kann gezeigt werden, dass

$$W(E_{i+1}) = W(E_i) - W(M_i). \tag{7}$$

  Somit wird das Gesamtkantengewicht des gröberen Graphen um das Gewicht des $Matching$ verringert. Durch die Auswahl eines maximalen Matchings M_i, dessen Kanten ein großes Gewicht haben Gewicht haben, können wir das Kantengewicht des gröberen Graphen um einen größeren Betrag verringern. Da er ein geringeres Kantengewicht hat, hat auch einen kleineren Kantenschnitt.
  Anstatt jedoch einen Knoten u zufällig mit einem seiner benachbarten nicht gematchten Knoten zu matchen, wird u mit dem Knoten v so gematcht, dass das Gewicht der Kante (u, v) das Maximum über alle gültigen inzidenten Kanten ist (schwerere Kante). Die Komplexität der Berechnung eines HEM ist $O(|E|)$.

6.1.2 Partitionierungsphase :

Die zweite Phase eines Multilevel-Algorithmus berechnet eine qualitativ hochwertige Bisektion (d. h. einen kleinen Kantenschnitt) P_m des groben Graphen $G_m = (V_m, E_m)$, so dass jeder Teil ungefähr die Hälfte des Knotengewichts des ursprünglichen Graphen enthält. Da bei der Vergröberung die Gewichte der Knoten und Kanten des vergröberten Graphen so eingestellt ist, dass sie die Gewichte der Knoten und Kanten des feineren Graphen widerspiegeln, enthält G_m Informationen, um auf intelligente Weise die balancierte Partitionierung und die kleinen Kantenschnitt-Anforderungen durchzusetzen.
Eine Partition von G_m kann mit verschiedenen Algorithmen erhalten werden, wie z.B [3]:

- spektrale Bisektion SB

- geometrische Bisektion (wenn Koordinaten verfügbar sind)
- kombinatorische Methoden

Da die Größe des gröberen Graphen G_m klein ist (d.h. $|V_m|< 100$), nimmt dieser Schritt nur wenig Zeit in Anspruch. Es wird nur einen Algorithmus in diesem Abschnitt vorgestellt und zwar **Graph growing partitioning algorithm (GGP)**. Eine einfache Methode zur Bisektion des Graphen und sie funktioniert wir folgt:
Man startet bei einem beliebigen Knoten und bildet eine Region um ihn herum, bis die Hälfte der Knoten oder die Hälfte des gesamten Knotengewichtes einbezogen ist. Die Qualität des *GGP* ist abhängig von der Wahl eines Knotens, von dem aus das Wachstum des Graphen beginnt, und verschiedene Startknoten ergeben unterschiedliche Kantenschnitte. Um dieses Problem teilweise zu lösen, wählen wir zufällig 10 Eckpunkte und lassen 10 verschiedene Regionen wachsen. Der Versuch mit dem kleineren Kantenschnitt wird als die Partition ausgewählt. Diese Partition wird dann weiter verfeinert, indem sie als Eingabe für die KL [3].

6.1.3 Entgröberungsphase (*Uncoarsening*) :

Während der Vergröberungsphase wird die Partition P_m des vergröberten Graphen G_m auf den ursprünglichen Graphen zurückprojiziert, indem man durch die Graphen $G_{m-1}, G_{m-2}, \ldots, G_1$ geht. Da jeder Knote con G_{i+1} eine bestimmte Teilmenge von Knoten von G_i enthält, erhält man P_i aus P_{i+1} durch einfaches Zuordnen der Menge der Knoten V_i^v, die mit $v \in G_{i+1}$ kollabiert sind, zu der Partition $P_{i+1}[v]$ (d.h $P_i[u] = P_{i+1}[v] \quad \forall u \in V_i^v$).
Der grundlegende Zweck eines Algorithmus zur Verfeinerung der Partition besteht darin, zwei Teilmengen von Knoten auszuwählen, eine aus jedem Teil, so dass beim Tausch die resultierende Partition einen kleineren Kantenschnitt aufweist. Wenn A und B die beiden Teile der Bisektion sind, wählt ein Verfeinerungsalgorithmus $A^{'} \in A$ und $B^{'} \in B$, so dass $A \setminus A^{'} \cup B^{'}$ und $B \setminus B^{'} \cup A^{'}$ eine Bisektion mit einem kleineren Kantenschnitt ist. Ein solcher Algorithmen ist $KL - refinement$ [3].
Der $KL - Algorithmus$ tauscht inkrementell Knoten zwischen den Partitionen einer Bisektion aus, um den Kantenschnitt der Partitionierung zu reduzieren, bis die Partitionierung ein lokales Minima erreicht. Eine Partitionierung hat ein lokales Minimum, wenn die Verschiebung eines beliebigen Knotens von einem Teil in den anderen nicht zu einer Verbesserung vom Kantenschnitt führt. Eine häufig verwendete Variante des KL-Algorithmus zur Verfeinerung der Bisektion stammt von **Fiduccia und Mattheyse**. Insbesondere berechnet diese Variante des KL-Algorithmus für jeden Knoten v den *Gain*, Der Gain ist die Verringerung des Kantenschnitts, die durch das Verschieben von v in die die andere Partition. Diese Knoten werden in zwei Prioritätswarteschlangen eingefügt, eine für jede Partition, je nach ihren *Gains*. Zum Beginn sind alle Knoten unlocked, d.h. sie sind frei in die andere Partition zu wechseln. Der Algorithmus wählt iterativ einen nicht unlocked Knoten v mit dem größten Gain aus einer der beiden Prioritätswarteschlangen aus und verschiebt ihn in die andere Partition. Wenn

ein Knoten v verschoben wird, wird er gelocked und der Gain von den benachbarten Knoten von v wird aktualisiert. Nach jeder Verschiebung eines Knotens speichert der Algorithmus auch die Größe des Schnittes, die an diesem Punkt erreicht wurde. Ein einzelner Durchlauf des Algorithmus endet, wenn es keine unlocked Knoten mehr gibt (d.h. wenn alle Knoten verschoben wurden). Dann werden die gespeicherten Schnittgrößen überprüft, der Punkt, an dem der minimale Schnittgröße erreicht wurde, wird ausgewählt, und alle Knoten, die nach diesem Punkt verschoben wurden, werden in ihre ursprüngliche Partition zurückgeschoben. Dies wird nun als den Anfang der Partitionierung für den nächsten Durchlauf des Algorithmus [2].

7 Anwendungen

Consensus Funktionen ermöglichen eine Vielzahl von neuen Ansätzen für verschiedene Probleme. Ein dieser Ansatz ist die *Segmentierungskombination* [7]. In diesem Abschnitt wird nur der Algorithmus von [7] beschrieben. Die Grundlage unseres Kombinationsalgorithmus für die Mehrfachsegmentierung Algorithmus ist der Random-Walker-Algorithmus zur Bild Segmentierung. Der Random-Walker-Algorithmus wird für einen ungerichteten Graphen $G = (V, E, w)$ formuliert, in dem jedes Pixel p_i einen entsprechenden Knoten $v_i \in V$ hat. Jede Kante $e_{ij} \in E$ hat ein Gewicht w_{ij}, das die Ähnlichkeit zwischen den benachbarten Pixeln v_i und v_j (in 4-Nachbarschaft). Zur Entwicklung eines Segmentierungs Ensemble-Kombinationsalgorithmus auf Basis des Random Walker benötigen wir drei Komponenten:

- Graph Generation: In unserem Zusammenhang soll das Gewicht w_{ij} angeben, wie wahrscheinlich die beiden Pixel p_i und p_j zur gleichen Bildregion gehören. **something missed**. Wir definieren also die Gewichtungsfunktion als Gaußsche Gewichtung:

$$w_{ij} = \exp(-\beta \cdot (1 - \tfrac{n_{ij}}{N})) \tag{8}$$

  wobei β ein freier Parameter des Algorithmus ist. Niedrige Kanten Gewichte weisen auf hohe Wahrscheinlichkeiten für eine Regionsgrenze zwischen zwei benachbarten Pixeln und verhindern, dass ein zufälligen Walker, der diese Grenzen überschreitet.
- Seed Region Generation: Dieser Schnitt wird in zwei Teile geteilt. In dem ersten bauen wir einen neuen Graphen G^* unter Beibehaltung der Kanten nur mit dem Gewicht $w_{ij} = 1$ und entfernen alle anderen Kanten. In diesem werden im Wesentlichen die Kanten zwischen zwei benachbarten Knoten (Pixeln) beibehalten, die mit hoher Wahrscheinlichkeit zur selben Region gehören.
- Segmentation Ensemble Combination

Literatur

[1] Tahani Alqurashi und Wenjia Wang. "Clustering ensemble method". In: *International Journal of Machine Learning and Cybernetics* 10.6 (2019), S. 1227–1246.

[2] Ranjit Jhala und Kenneth L McMillan. "A practical and complete approach to predicate refinement". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2006, S. 459–473.

[3] George Karypis und Vipin Kumar. "A fast and high quality multilevel scheme for partitioning irregular graphs". In: *SIAM Journal on scientific Computing* 20.1 (1998), S. 359–392.

[4] Kilian. *Der Clustering Guide: Definition, Methoden und Beispiele*. 2022. URL: `https://www.kobold.ai/clustering-guide/`.

[5] Alexander Strehl und Joydeep Ghosh. "Cluster ensembles—a knowledge reuse framework for combining multiple partitions". In: *Journal of machine learning research* 3.Dec (2002), S. 583–617.

[6] Sandro Vega-Pons und José Ruiz-Shulcloper. "A survey of clustering ensemble algorithms". In: *International Journal of Pattern Recognition and Artificial Intelligence* 25.03 (2011), S. 337–372.

[7] Pakaket Wattuya u. a. "A random walker based approach to combining multiple segmentations". In: *2008 19th International Conference on Pattern Recognition*. IEEE. 2008, S. 1–4.