technische universität
dortmund

Masterarbeit

Kernel k-means Methoden zur spektralen Clusteranalyse von Graphen

Lukas Pradel
4. Februar 2015

Gutachter:
Prof. Dr. Christian Sohler
Dipl.-Inf. Melanie Schmidt

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl II - Effiziente Algorithmen und Komplexitätstheorie
http://ls2-www.cs.tu-dortmund.de/

Hier kommt eine Danksagung hin!

Inhaltsverzeichnis

1 Einleitung

Die Clusteranalyse beschäftigt sich mit Verfahren, die Ähnlichkeitsstrukturen in (großen) Datenmengen identifizieren. Der englische Begriff *cluster* lässt sich in diesem Kontext am ehesten mit Gruppe oder Ansammlung übersetzen. Dem Gebiet liegt die Annahme zu Grunde, dass in gegebenen Datenmengen typischerweise Strukturen existieren, die mit einem geeigneten Ähnlichkeitsmaß erkannt werden können. Diese grundlegende Annahme ist insofern plausibel, als dass die zu analysierenden Datenmengen in der Praxis in der Regel aus einem konkreten Anwendungsfall gesammelt oder aufgezeichnet werden, in dem bestimmte Muster oder sich wiederholende Prozesse existieren. Bei der Analyse von Daten aus natürlichen Prozessen, sind diese beispielsweise oft gemäß einer bestimmten Wahrscheinlichkeitsverteilung verteilt. Sind die vorhandenen Strukturen einmal erkannt, kann der gesamte Datensatz in Gruppen oder *Cluster* gegliedert werden, sodass innerhalb eines Clusters große Ähnlichkeit zwischen Objekten besteht, während Objekte, die unterschiedlichen Clustern zugeordnet sind, geringe Ähnlichkeit aufweisen. Anhand der Cluster, die mit Verfahren der Clusteranalyse gebildet wurden, lassen sich dann häufig aufschlussreiche Aussagen über die Eingabedaten treffen, die beim bloßen Anblick der gesamten unstrukturierten Datenmenge nicht möglich gewesen wären. In manchen Anwendungen ist es auch hilfreich, die Teilmengen der Eingabe zu verarbeiten, wenn zusätzliche Informationen oder Annahmen über die Eingabe vorliegen.

Wir betrachten dazu einleitend ein einfaches Beispiel der Gegenwart: die Analyse von Strukturen in sozialen Netzwerken. Die Eingabeobjektmenge besteht in diesem Fall aus Objekten, die Informationen über Beziehungen und Aktivitäten zwischen Nutzern des sozialen Netzwerks enthalten. Mit Hilfe der Clusteranalyse können beispielsweise Gruppen von Benutzern identifiziert werden, die viel miteinander interagieren und somit eine soziale Community, also eine Gruppe von Freunden oder Bekannten, bilden. In diesem Beispiel kann es auch aufschlussreich sein, sogenannte „schwarze Löcher“ zu identifizieren, also Benutzer, die eine große Menge von eingehenden Aktivitäten zahlreicher anderer Benutzer haben, selbst aber nur selten mit einzelnen Benutzern interagieren. Bei Benutzern mit diesem Verhalten handelt es sich häufig um Prominente mit einer großen Fangemeinde. Auch Benutzer, die keine eingehenden oder ausgehenden Aktivitäten haben, können interessant sein. Dabei handelt es sich um „Einzelgänger“, die gegebenenfalls gezielt angesprochen werden können. Unabhängig vom konkreten verfolgten Ziel wird schnell klar, dass entsprechende Aussagen gerade bei Netzwerken mit einer großen Nutzeranzahl, wie

beispielsweise Facebook, nur dann möglich sind, wenn Verfahren der Clusteranalyse eingesetzt werden. Würde man lediglich die unstrukturierte gesamte Datenmenge betrachten, ließen sich nur sehr schwer Informationen gewinnen, die keine Aussagen über das globale Netzwerk treffen. Es wird ebenso klar, dass es nötig ist, Verfahren zu entwickeln, die innerhalb der Grenzen gegenwärtiger Hardware Ergebnisse liefern, wenn die Eingabedatenmenge sehr große bis gigantische Größenordnungen annimmt.

In dieser Arbeit beschäftigen wir uns mit einem weit verbreiteten Clusteringmodell, in dem das Ähnlichkeitsmaß die (quadrierte) geometrische Distanz ist. Die Eingabe besteht demnach aus einer Menge P von Punkten aus dem euklidischen Raum $\mathbb{R}^d$. Neben der Punktmenge P enthält unsere Eingabe zusätzlich eine positive ganze Zahl k, welche die Anzahl an Clustern angibt, in welche die Punkte unterteilt werden sollen. Dieses Modell ist unter dem Namen „k-means-Problem“ bekannt. Den Algorithmen für das k-means-Problem, die wir betrachten wollen, ist gemein, dass sie eine initiale Auswahl von Clustern und zugehörigen *Zentren* iterativ weiter verbessern, bis eine zumindest lokal optimale Lösung ermittelt wurde. Der Idee, jedem Cluster ein Zentrum zuzuordnen, liegt die nützliche Erkenntnis zugrunde, dass die optimale Lösung des 1-means-Problems der geometrische Zentroid der Punktmenge ist. Die eigentliche Aufgabe besteht dann darin, die Summe der (quadrierten) Distanzen der Punkte eines Clusters zu ihrem Clusterzentrum zu minimieren.

Da das k-means-Problem bekanntermaßen schon für $k = 2$ NP-hart ist [ADHP09], sind für praktische Zwecke insbesondere Heuristiken, also Algorithmen ohne feste Schranken für Laufzeit oder Qualität der Lösung) und Approximationsalgorithmen, die eine nicht-optimale Lösung berechnen, welche jedoch garantiert nur um einen gewissen Faktor von der optimalen Lösung abweichen, interessant. Wir betrachten in dieser Hinsicht die Konstruktion von *Kernmengen*. Bei Eingabe einer Punktmenge P für das k-means-Problem handelt es sich bei einer Kernmenge um eine kleinere gewichtete Punktmenge S mit der Eigenschaft, dass für jede Wahl von k Clusterzentren die Summe der quadrierten Distanzen der Punkte in P zu den Zentren in etwa so groß ist, wie die Summe der quadrierten Distanzen der Punkte in S zu den Zentren. Effektiv ist eine Kernmenge also ein hinsichtlich der Clusteringzielfunktion repräsentatives, (teilweise deutlich) kleineres Abbild der ursprünglichen Punktmenge. Üblicherweise bieten Kernmengenkonstruktionen Garantieren einer $(1 + \epsilon)$-Approximation, das heißt, dass ein k-means-Clustering auf der Kernmenge höchstens um einen Faktor $(1 + \epsilon)$ schlechter ist, als das selbe Clustering auf der Ursprungsmenge P. Neben der Approximation des k-means-Problems können

Kernmengen auch ein effektives Mittel im Umgang mit großen Datenmengen sein. Wenn die Größe der Kernmenge S nur ein kleiner Anteil an der Größe der von P ist, können Clustering-Algorithmen auf S deutlich schneller ausgeführt werden als auf P.

Schließlich betrachten wir in dieser Arbeit noch eine weitere Modellierung von Clusteringproblemen, die mit der geometrischen Auffassung verwandt ist. Für einige Anwendungen ist es sinnvoll, die zu clusternden Objekte als Knoten eines gewichteten Graphen zu betrachten. Das Ähnlichkeitsmaß ist dann nicht mehr die euklidische Distanz sondern das Gewicht der Kanten zwischen den Objekten beziehungsweise Knoten. Stellt man sich den gezeichneten Graphen vor, kommt das Kantengewicht der „Distanz“ zwischen den Knoten gleich. Bei der Clusteranalyse von Graphen können wir unter anderem auf diverse Ergebnisse aus der Graphentheorie zurückgreifen und zudem Techniken aus der linearen Algebra anwenden, wie wir später aufzeigen werden.

Anwendungen. Die Clusteranalyse war in der Vergangenheit bereits Gegenstand intensiver Forschung, die bis heute regelmäßig neue theoretische Erkenntnisse liefert. Es handelt sich aber um ein Gebiet mit hoher praktischer Relevanz, da Clustering in zahlreichen Anwendungen entweder selbst durchgeführt werden muss, oder als Teilproblem in einem übergeordneten Kontext auftritt. Wir wollen an dieser Stelle einen kurzen und sicherlich unvollständigen Überblick über einige wichtige Anwendungen der bisher vorgestellten Clusteringmodelle geben.

Eine gegenwärtige Anwendung für k-means-Clustering haben wir mit der Analyse sozialer Netzwerke bereits erwähnt. Eine weitere Anwendung, die gerade heute im Bereich der Online-Shops relevant ist, findet das k-means-Clustering in der Marktforschung. Hier sollen Informationen über das Einkaufsverhalten von Kunden genutzt werden, um diese beispielsweise den Marktsegmenten nach zu unterteilen. Je nachdem, wie detailliert die gesammelten Informationen sind, können diese auch genutzt werden, um Produktplatzierung zu optimieren, die Nachfrage nach neuen Produkten abzuschätzen, oder potenzielle neue Märkte zu erkennen. Neben den wirtschafts- und sozialwissenschaftlichen Anwendungsfällen, kommt die Clusteranalyse auch häufig in geographischen und geologischen Zusammenhängen zum Einsatz. Konkret werden hier chemische Eigenschaften von gesammelten Proben geclustert, um geographische Profile der untersuchten Substanzen zu erstellen.

Kernmengen werden aktuell hauptsächlich in drei Anwendungen eingesetzt. Wie bereits erwähnt bieten viele Kernmengenkonstruktionen eine garantierte Approxi-

mationsgüte, das heißt das anhand der Kernmenge berechnete Clustering weicht für jede beliebige Wahl der Zentren aus $\mathbb{R}^d$ um beispielsweise einen Faktor von höchstens $1 + \epsilon$ von einem Clustering mit den selben Zentren auf der Ursprungsmenge ab. Man spricht in diesem Zusammenhang auch von *starken* Kernmengen. Ist diese Eigenschaft erfüllt, können wir mit einer Kernmenge, die eine deutlich kleinere Größe hat, als die Ursprungsmenge, für Approximationsalgorithmen für das k-means-Problem einen signifikanten *Speedup*, also eine Beschleunigung der Ausführungszeiten, erzielen. Dabei muss selbstverständlich beachtet werden, dass die Konstruktion der Kernmenge selbst ebenfalls Zeit in Anspruch nimmt und nicht zu aufwändig sein darf, wenn ein Speedup beabsichtigt wird.

Ein relativ neues Modell für Algorithmen sind sogenannte *Datenströme*, bei denen die Eingabe nicht initial vollständig vorliegt, sondern kontinuierlich in einem Datenstrom geliefert wird. Algorithmen für Datenströme, also *Datenstromalgorithmen* dürfen die Daten des Stroms nur genau einmal einlesen und sind zusätzlich polylogarithmisch bezüglich der Eingabegröße platzbeschränkt, das heißt es ist nicht möglich die gesamte Eingabe zu protokollieren. Kernmengen können bei Clustering in Datenströmen insofern hilfreich sein, als dass in bestimmten Intervallen für einen Teil der Eingabe eine Kernmenge berechnet wird. Sobald der Speicherplatzverbrauch zu groß wird, wird eine einzelne Kernmenge für die Vereinigung aller bisher berechneten Kernmengen bestimmt und mit dieser weitergearbeitet. Die Anwendung von Kernmengen im Bereich von Datenstromalgorithmen ist Gegenstand gegenwärtiger Forschung und hat bereits erste Ergebnisse erbracht, wie wir später noch sehen werden.

Schließlich können wir Kernmengen mit der selben Idee einsetzen, um Clusteringalgorithmen zu parallelisieren oder zu verteilen. Dazu partitionieren wir die Eingabepunktmenge, berechnen auf den Teilen verteilt oder parallel Kernmengen, übermitteln die berechneten Kernmengen an eine zentrale Instanz und vereinen diese dort zu einer gesamten Kernmenge.

Die Clusteranalyse von Graphen findet ihre vielleicht wichtigste Anwendung in der Bildsegmentierung. Sie wird in diesem Kontext eingesetzt, um bei digitalen Bildern Kanten und insbesondere Objekte zu erkennen. Die Kantenerkennung ist eine fundamentale Technik in der digitalen Bildverarbeitung, die für detaillierte Verfahren als Grundlage nötig ist. Das Erkennen von Objekten macht beispielsweise die Erkennung von Gesichtern, Personen oder Orten auf Fotos möglich.

Darüber hinaus ergibt sich eine natürliche Anwendung in der Identifikation relevanter Strukturen und insbesondere der Konnektivitätsanalyse in Netzwerken. Auch die Entdeckung von häufigen Anrufmustern in der Telekommunikation kann hier

von Interesse sein. Schließlich kann es in Netzwerken mit dynamischer Topologie, wie beispielsweise *ad-hoc-Netzwerken*, wie sie beispielsweise in intelligenten Umgebungen mit vielen wechselnden mobilen Geräten vorkommen. Lokales Clustering wird hier unter anderem eingesetzt, um Routing-Algorithmen zu verbessern.

Zudem kann die Clusteranalyse von Graphen eingesetzt werden, um Caching-Mechanismen in Datenbanken zu optimieren. Konkret werden in Datenbanksystemen die einzelnen Einträge in Form von *Seiten* auf dem Speichermedium abgelegt, von denen einige im Hauptspeicher gecached werden. Ein gutes Clustering führt in diesem Kontext dazu, dass das Laden eines Clusters, dem ein angefragtes Datum zugeordnet ist, zudem relevante Daten liefert, die für ähnliche zukünftige Anfragen erforderlich sind.

Verwandte Arbeiten. Blabla

2 Grundlegende Definitionen und Algorithmen

In diesem Kapitel definieren wir die für unsere Zwecke relevanten Begriffe im Kontext der Clusteranalyse und führen die wichtigen grundlegenden Algorithmen ein, deren Ideen für uns im Folgenden noch von Bedeutung sein werden. Wir gehen dabei nach Themengebieten geordnet vor: In Abschnitt 2.1 skizzieren wir kurz das Themengebiet der Clusteranalyse, definieren die üblichen Zielfunktionen und stellen zwei bedeutende Algorithmen vor. Abschnitt 2.2 führt kurz in die Graphentheorie sowie die Clusteranalyse von Graphen ein. In diesem Abschnitt werden wir zudem verschiedene Optimierungskritierien für die Clusteranalyse von Graphen herausstellen. Schließlich fassen wir in Abschnitt 2.3 die wichtigsten Methoden und Algorithmen aus dem Bereich der spektralen Clusteranalyse zusammen und stellen zudem die wesentlichen Konzepte von Kernel-Methoden vor.

2.1 Clustering und k-means

Clusteranalyse oder „Clustering“ beschäftigt sich mit der Einteilung von Objekten in Gruppen („Cluster“), sodass sich die Objekte innerhalb eines Clusters gemäß eines bestimmten Optimierungskriteriums ähnlich sind und von Objekten eines anderen Clusters unterscheiden. Es existieren zahlreiche grundsätzlich verschiedene Ansätze, Clusteringprobleme zu lösen. Wir beschränken uns in dieser Arbeit auf *partitionierende* Clusteringprobleme und -verfahren. Bei diesen soll eine Menge von Objekten, welche der erste Teil der Eingabe ist, gemäß einer Cluster-Zielfunktion möglichst optimal in genau k Cluster unterteilt werden. Dabei ist k der ganzzahlige zweite Teil der Eingabe, das heißt, die Anzahl an Clustern muss bei diesen Verfahren vorab festgelegt werden. Die initial gewählten k Zentren werden dann iterativ verschoben, bis ein gewisses Abbruchkriterium erfüllt ist.

Für die Zielfunktion, welche die Nähe oder Ferne von Punkten zueinander quantifiziert, sind bei Eingabepunkten aus $\mathbb{R}^d$ Metriken naheliegend. Intuitiv ist dabei die euklidische Distanz, auf der die beiden bekanntesten Clustering-Problemstellungen basieren. Wir notieren die euklidische Distanz zwischen zwei Punkten $x, y \in \mathbb{R}^d$ mit $\|x - y\|$.

Definition 2.1.1 (k-median und k-means). Sei $P \subset \mathbb{R}^d$ und $k \in \mathbb{N}^+$. Das k-median-Problem besteht darin, eine Menge von k (Cluster-)*Zentren* $C = \{c_1, \ldots, c_k\}$ mit $c_i \in \mathbb{R}^d$ zu finden, sodass der folgende Term minimal wird:

$$\sum_{p \in P} \min_{c \in C} \|p - c\|$$

Das k-means-Problem unterscheidet sich nur darin, dass bei diesem die Summe der *quadrierten* euklidischen Distanzen zum jeweils nächstgelegenen Zentrum minimiert werden soll, das heißt, dass der folgende Term minimiert werden soll:

$$\sum_{p \in P} \min_{c \in C} \|p - c\|^2$$

Beim *gewichteten* k-means-Problem werden den Eingabepunkten zusätzlich mit einer Funktion $w : P \to \mathbb{R}$ Gewichte zugewiesen. Die zu minimierende Zielfunktion lautet dann entsprechend

$$\sum_{p \in P} \min_{c \in C} w(p)\|p - c\|^2$$

Sowohl das k-Median-Problem [MS84] als auch das k-means-Problem [ADHP09] sind optimal NP-schwer lösbar. Typischerweise werden zur Lösung daher approximative oder heuristische Algorithmen eingesetzt. Die bekannteste und bis heute sehr erfolgreiche Heuristik für das k-means-Problem ist der Algorithmus von Lloyd [Llo82]. Wegen seiner großen Popularität wird der Algorithmus häufig auch als „k-means-Algorithmus“ bezeichnet. Der Algorithmus wählt initial k zufällige Punkte aus der Eingabemenge als initiale Clusterzentren. Darüber hinaus existieren zahlreiche weitere Methoden zur initialen Wahl der Zentren, siehe dazu beispielsweise [CKV13]. Anschließend wird jedem Punkt das am nächsten gelegene Zentrum zugewiesen. Dadurch entstehen die initialen Cluster mit ihren jeweiligen Zentren. Im zweiten Schritt wird das neue Zentrum eines jeden Clusters als der geometrische Zentroid des Clusters gewählt. Die Wahl des geometrischen Zentroiden als neues Zentrum ist für das 1-means-Problem optimal [ORSS12]. Die Zuweisung von Punkten zum nächstgelegenen Clusterzentrum und die Berechnung der neuen Zentroiden werden solange alterniert, bis die Lösung konvergiert, also wenn sich die Zuordnungen der Punkte nicht mehr ändern. In der Praxis wird gelegentlich auch nach einer festen Anzahl von Iterationen terminiert.

Algorithmus 1: Algorithmus von Lloyd

Eingabe: $: P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+$

Ausgabe: $:$ k-means-Clustering von P

1 Wähle zufällig k Zentren $c_1^{(0)}, \ldots, c_k^{(0)}$ aus P oder $\mathbb{R}^d$
2 $S_i^{(0)} \leftarrow \{p \in P : \|p - c_i^{(0)}\|^2 \leq \|p - c_{i'}^{(0)}\|^2 \, \forall \, i' \in \{1, \ldots, k\}\}$
3 **repeat**
4 $\quad c_i^{(t)} \leftarrow \frac{1}{|S_i^{(t-1)}|} \sum_{p_j \in S_i^{(t-1)}} p_j$
5 $\quad S_i^{(t)} \leftarrow \{p \in P : \|p - c_i^{(t)}\|^2 \leq \|p - c_{i'}^{(t)}\|^2 \, \forall \, i' \in \{1, \ldots, k\}\}$
6 **until** $S_i^{(t)} = S_i^{(t-1)}$

Die asymptotische Laufzeit des Algorithmus beträgt $\mathcal{O}(nkdi)$, wobei i die Anzahl an durchgeführten Iterationen ist. Wenn der Algorithmus konvergiert und nicht durch eine feste Anzahl von Iterationen terminiert wird, wurde ein lokales Optimum gefunden, welches jedoch im Allgemeinen kein globales Optimum oder eine Approximation eines globalen Optimums ist.

Der Algorithmus k-means++ [AV07] sieht eine alternative Wahl der initialen Clusterzentren in Lloyds Algorithmus vor: er wählt diese auf einfache, aber dennoch geschickte Art und Weise und führt anschließend die übrigen Schritte von Lloyds Algorithmus aus. Dazu wird zunächst ein einzelnes Clusterzentrum c_1 zufällig gleichverteilt aus der Eingabe-Punktmenge P gewählt. Alle weiteren Clusterzentren werden sukzessive nach der folgenden Vorschrift gewählt, bis insgesamt k Zentren gewählt wurden. Im Weiteren bezeichnen wir mit $D(x)$ für einen Punkt x aus der Eingabe-Punktmenge P die geringste Distanz von x zum nächstgelegenen bereits gewählten Zentrum. In jeder Iteration wird als nächstes Zentrum c_i der Punkt $x' \in P \setminus \{c_1, \ldots, c_{i-1}\}$ mit Wahrscheinlichkeit $\frac{D(x')^2}{\sum_{x \in P} D(x)^2}$ gewählt.

Algorithmus 2: k-means++

Eingabe: : $P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+$

Ausgabe: : k initiale Clusterzentren für P

1 Wähle c_1 zufällig gleichverteilt aus P
2 **for** $i \leftarrow 2$ **to** k **do**
3 Wähle den Punkt $x' \in P \setminus \{c_1, \ldots, c_{i-1}\}$ als Zentrum c_i mit Wahrscheinlichkeit $\frac{D(x')^2}{\sum_{x \in P} D(x)^2}$
4 Führe Lloyds Algorithmus mit den initialen Clusterzentren $c_1, \ldots, c_k$ aus.

Die k Zentren, die von k-means++ ausgewählt werden, sind eine $\mathcal{O}(\log k)$-Approximation für das k-means-Problem, die durch die anschließende Ausführung von Lloyds Algorithmus noch zu einem lokalen Optimum verbessert werden.

Im nächsten Abschnitt betrachten wir das Clusteringproblem auf Graphen, das wir im später noch genauer untersuchen wollen.

2.2 Graphen und Clusteranalyse von Graphen

Der *Graph* ist in der Informatik eine vielseitig einsetzbare und gut erforschte Datenstruktur. Wir können diverse Clusteringprobleme nicht nur für eine Eingabepunktmenge formulieren, sondern auch für eine Eingabe, die aus einem Graphen besteht. Wir beginnen mit der folgenden Definition des Graphen.

Definition 2.2.1 (Graph). Sei V eine endliche Menge und $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$. Dann heißt das Tupel $G = (V, E)$ ein (endlicher) *Graph* mit *Knotenmenge* V und *Kantenmenge* E. Ist $e = \{u, v\} \in E$, dann sagen wir, dass die Kante e des Graphen G die Knoten u und v *verbindet*. In diesem Fall sind u und v die *Endknoten* von e. Ein Knoten $u \in V$ und eine Kante $e \in V$ heißen *inzident* genau dann, wenn $u \in e$. Wir sagen, dass u und ein weiterer Knoten $v \in V$ *adjazent* sind genau dann, wenn es eine Kante $e' = \{u, v\}$ in E gibt. Typischerweise bezeichnet $n = |V|$ die *Knotenzahl* von G und $m = |E|$ die *Kantenzahl* von G.

Bei einem 3-Tupel $G' = (V', E', w')$ mit $w' : E \to \mathbb{N}$ spricht man von einem *gewichteten* Graphen, dessen Kanten über ein Gewicht verfügen, das von der Gewichtsfunktion w' abgebildet wird.

Für die konkrete Datenhaltung von Graphen haben sich im Wesentlichen zwei Ansätze als praktikabel erwiesen: Bei den sogenannten *Adjazenzlisten* wird für jeden Knoten v im Graphen eine Liste Adj_v gehalten, in der für jeden zu v inzidenten Knoten ein Eintrag in Adj_v enthalten ist, der den entsprechenden adjazenten Knoten referenziert, sowie gegebenenfalls das Gewicht der Kante zwischen den beiden Knoten. Alternativ wird in einer *Adjazenzmatrix* Adj^G der Größe $|V| \times |V|$ an der Stelle $Adj^G_{u,v}$ das Gewicht der Kante zwischen den Knoten u und v eingetragen, sofern die beiden Knoten durch eine Kante miteinander verbunden sind. Anderenfalls wird zumeist -1 oder 0 eingetragen. Bei ungewichteten Graphen wird dementsprechend lediglich 0 oder 1 eingetragen.

Wenn die Eingabe keine Punktmenge, sondern ein (gewichteter) Graph ist, können wir die Ideen der bereits vorgestellten Verfahren weiterhin anwenden. Wir interessieren uns in diesem Fall für die „Ähnlichkeit“ von Knotenmengen im Graphen.

Definition 2.2.2 (Graph-Schnitt). Sei $G = (V, E, w)$ ein gewichteter Graph. Ein *Schnitt* $C = (S, T)$ von G ist eine Partitionierung von V in die beiden Mengen S und T, das heißt, dass $V = S \cup T$ und gleichzeitig $S \cap T = \emptyset$. Die *Schnittmenge* von C sind die Kanten in E, die einen Endpunkt in S und den anderen Endpunkt in T haben, das heißt formal ist die Schnittmenge definiert als

$$\{(u, v) \in E \mid u \in S, v \in T\}.$$

Das Gewicht oder der Wert eines Schnittes ist die Summe der Kantengewichte der Schnittmenge. Wir verwenden die folgende Notation:

$$w_{cut}(S, T) = \sum_{u \in S, v \in T} w((u, v))$$

Falls der Graph in Form einer Adjazenzmatrix Adj^G vorliegt, lautet die Berechnungsvorschrift entsprechend:

$$w_{cut}(S,T) = \sum_{u \in S, v \in T} Adj^G_{u,v}$$

Für eine direkte Analogie zum k-means-Problem wäre ein Schnitt- beziehungsweise Partitionierungs-Begriff wünschenswert, der eine k-fache Partitionierung der Knotenmenge erlaubt. Diese lautet wie folgt.

Definition 2.2.3 (k-Graphpartitionierung). Sei $G = (V, E, w)$ ein gewichteter Graph. Für zwei Mengen $A, B \subseteq V$ definieren wir:

$$w(A,B) = \sum_{u \in A, v \in B} w((u,v))$$

Das k-Graphpartitionierungsproblem besteht darin, eine Partitionierung der Knotenmenge V in k disjunkte Teilmengen $V_1, \ldots, V_k$ mit $\bigcup_{i \in \{1,\ldots,k\}} V_i = V$ zu ermitteln, sodass sich die Knoten innerhalb einer Partition bezüglich einer Ähnlichkeitsrelation möglichst ähnlich sind und die Knoten unterschiedlicher Partitionen bezüglich der Ähnlichkeitsrelation möglichst stark voneinander unterscheiden [KL70].

Beim Clustering von Punktmengen lagen für die Unähnlichkeitsrelation Metriken nahe, im Falle der Graphpartitionierung existiert eine Reihe von Optimierungskriterien, von denen wir im Folgenden die verbreitetsten einführen.

1. **Ratio Association**. Bei der Ratio Association sollen die Intra-Clusterabstände relativ zur jeweiligen Clustergröße maximiert werden:

$$\max \sum_{c=1}^{k} \frac{w(V_c, V_c)}{|V_c|}$$

2. **Ratio Cut**. Beim Ratio Cut wird das Gewicht des Schnitts zwischen jeweils einem Cluster und allen anderen Punkten im Graphen minimiert:

$$\min \sum_{c=1}^{k} \frac{w(V_c, V \setminus V_c)}{|V_c|}$$

3. **Kernighan-Lin**. Bei dem in [KL70] vorgestellten Optimierungskriterium werden die Intra-Clusterabstände ähnlich der Ratio Association minimiert, allerdings müssen alle Partitionen hier zusätzlich die selbe Größe haben. Die

hier vorgestellte Zielfunktion ist von 2 auf k Partitionen verallgemeinert:

$$\min \sum_{c=1}^{k} \frac{w(V_c, V \setminus V_c)}{|V_c|} \text{ s.t. } |V_c| = \frac{|V|}{k} \forall\, c \in \{1, \ldots, k\}$$

4. **Normalized Cut**. Ziel des Normalized Cut ist wie beim Ratio Cut die Minimierung des Schnitts von einem Cluster mit den übrigen Punkten im Graphen, jedoch relativ zum Schnitt des Clusters mit *allen* Knoten im Graphen. Der Normalized Cut oder kurz NCut ist in der Bild-Segmentierung recht verbreitet [SM00]. Wir betrachten hier ebenfalls die auf k Partitionen verallgemeinerte Zielfunktion.

$$\min \sum_{c=1}^{k} \frac{w(V_c, V \setminus V_c)}{w(V_c, V)}$$

Im nächsten Abschnitt betrachten wir zwei Techniken, mit denen Limitierungen hinsichtlich der Qualität der erzielten Clusterings sowie der Komplexität der Berechnungen verbessert werden können.

2.3 Kernel-Methoden und spektrales Clustering

Legt man die k-means Zielfunktion zu Grunde, können im Ursprungsraum $\mathbb{R}^d$ nur Cluster berechnet werden, die *linear separierbar* sind. Ein illustratives Beispiel im $\mathbb{R}^2$ sind zwei konzentrisch angeordnete ringförmige Punktmengen. Die intuitiv optimalen Cluster sind die beiden jeweiligen Ringe, es ist jedoch im $\mathbb{R}^2$ nicht möglich Clusterzentren anzugeben, die den zweidimensionalen Raum in Ringe partitionieren. Die Beschränkung liegt darin, dass wir die Partitionierung oder Separierung durch Hyperebenen vornehmen. Im Ursprungsraum können wir daher nur linear separierbare Cluster bestimmen. Wir betrachten nun ein Verfahren, mit dem sich diese Beschränkung umgehen lässt.

2.3.1 Kernel-k-means

Unsere fundamentale Motivation für diesen Abschnitt ist Covers Theorem [Cov65], welches informell zusammengefasst besagt, dass eine zufällige Eingabepunktmenge im d-dimensionalen Raum mit hoher Wahrscheinlichkeit in einen höherdimensionalen Raum (zum Beispiel der Dimension $D \gg d$) abgebildet linear separierbar ist.

Für Klassifizierungs- und insbesondere Clusteringprobleme benötigen wir sowohl im Ursprungsraum als auch im abgebildeten Raum ein algebraisches Maß für

Unähnlichkeit. Üblicherweise wird dabei eine auf *Skalarprodukten* basierende Metrik eingesetzt. Wir beziehen uns im Weiteren auf die folgende Definition von *euklidischen Vektorräumen*, also Vektorräumen mit einem reellen Skalarprodukt.

Definition 2.3.1 (Euklidischer Vektorraum). Sei V ein Vektorraum über $\mathbb{R}$. Ein reelles Skalarprodukt auf V ist eine Abbildung $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$, die den folgenden Eigenschaften genügt:

1. $\langle a, b \rangle = \langle b, a \rangle \, \forall \, a, b \in V$
2. $\langle \lambda a + \mu b, c \rangle = \lambda \langle a, c \rangle + \mu \langle b, c \rangle \, \forall \, a, b, c \in V$ und $\lambda, \mu \in \mathbb{R}$
3. $\langle a, a \rangle \geq 0$ und $\langle a, a \rangle = 0 \Leftrightarrow a = 0 \, \forall \, a \in V$

Ein $\mathbb{R}$-Vektorraum mit einem reellen Skalarprodukt heißt *euklidischer Vektorraum*.

Die k-means-Zielfunktion kann auf euklidische Vektorräume erweitert werden, indem wir für einen euklidischen Vektorraum $\mathcal{V}$ eine Norm $\|\cdot\| : \mathcal{V} \to \mathbb{R}$ über das Skalarprodukt gemäß der folgenden Abbildungsvorschrift für jedes $\chi \in \mathcal{V}$ definieren: $\|\chi\| = \sqrt{\langle \chi, \chi \rangle}$. Das auf Skalarprodukten basierte Analog zu Definition 2.1.1 des k-means-Problems lautet dann wie folgt:

Definition 2.3.2 (k-means in euklidischen Vektorräumen [Sch14]). Sei $\mathcal{V}$ ein euklidischer Vektorraum mit der Norm $\|\cdot\| : \mathcal{V} \to \mathbb{R}$ und sei $\mathcal{P} \subset \mathcal{V}$ sowie $k \in \mathbb{N}^+$. Das k-means-Problem in $\mathcal{V}$ besteht darin, eine Menge von k (Cluster-)Zentren $\mathcal{C} = \{c_1, \ldots, c_k\}$ mit $c_i \in V$ zu finden, sodass der folgende Term minimal wird:

$$\sum_{p \in \mathcal{P}} \min_{c \in \mathcal{C}} \|p - c\|^2$$

Die gewichtete Variante des k-means-Problems in euklidischen Vektorräumen lautet entsprechend wie folgt:

$$\sum_{p \in \mathcal{P}} \min_{c \in \mathcal{C}} w(p) \|p - c\|^2$$

Damit ist es nun möglich, ein intuitives Verfahren zum Bestimmen von Clusterings mit nicht-linearen Separatoren anzugeben. Wir bilden zunächst mit einer Abbildung $\phi : \mathcal{X} \to \mathcal{V}$ unsere Eingabepunkte aus ihrem Ursprungsraum $\mathcal{X}$ (also beispielsweise $\mathbb{R}^d$) auf einen D-dimensionalen (üblicherweise $D \gg d$) euklidischen Vektorraum $\mathcal{V}$ ab, und berechnen in diesem mit Lloyds Algorithmus oder einem anderen Clusteringverfahren ein Clustering, welches im Ursprungsraum $\mathcal{X}$ im Allgemeinen

nicht-linear separierten Clustern entspricht. Bei diesem Verfahren wird entsprechend der folgende Term minimiert:

$$\sum_{p \in P} \min_{c \in C} \|\phi(p) - c\|^2$$

Die Zentroid-Berechnung zur Bestimmung eines Clusterzentrums c_i im Sinne von Lloyds Algorithmus lautet dann

$$c_i = \frac{\sum_{p \in S_i} \phi(p)}{|S_i|}$$

wobei S_i die Menge aller dem Clusterzentrum c_i zugeordneten Punkte ist. Zunächst haben wir uns damit nicht-linear separierte Cluster „erkauft“, der Preis, den wir dafür zahlen, besteht jedoch in höherem Rechenaufwand, da wir einerseits die Punkte in den jeweils höherdimensionalen Raum abbilden müssen und andererseits die Berechnung der (euklidischen) Distanzen aufwändiger wird; diese benötigt im Ursprungsraum Zeit $\mathcal{O}(d)$ und im höherdimensionalen Raum $\mathcal{O}(D)$. Der Mehraufwand für die Distanzberechnung fällt dabei stärker ins Gewicht, da wir die Distanzberechnung beispielsweise bei Lloyds Algorithmus in jeder Schleifeniteration durchführen.

Dieses Problem wurde erstmals im Zusammenhang mit *Support Vector Machines* (kurz SVMs) gelöst [BGV92]. Wir übertragen die Lösung mittels des sogenannten *Kernel-Tricks* in den Clustering-Kontext und skizzieren diese kurz. Zunächst halten wir eine wichtige Beobachtung hinsichtlich der (quadrierten) euklidischen Distanz $\|\phi(p) - c\|^2$ zwischen dem abgebildeten Punkt und einem Clusterzentrum fest. Diese lässt sich wie folgt ausschließlich über Skalarprodukte berechnen (wobei auch hier wieder S_c die dem Clusterzentrum c zugeordnete Punktmenge ist):

$$\begin{aligned}
\|\phi(p) - c\|^2 &= \left\| \phi(p) - \frac{1}{|S_c|} \sum_{x \in S_c} \phi(x) \right\|^2 \\
&= \left\langle \phi(p) - \frac{1}{|S_c|} \sum_{x \in S_c} \phi(x), \phi(p) - \frac{1}{|S_c|} \sum_{x \in S_c} \phi(x) \right\rangle \\
&= \langle \phi(p), \phi(p) \rangle - \frac{1}{|S_c|} \sum_{x \in S_c} \langle \phi(p), \phi(x) \rangle - \frac{1}{|S_c|} \sum_{x \in S_c} \langle \phi(x), \phi(p) \rangle \\
&\quad + \frac{1}{|S_c|^2} \sum_{x,y \in S_c} \langle \phi(x), \phi(y) \rangle
\end{aligned} \tag{2.1}$$

Diese Beobachtung allein hilft uns noch nicht weiter, da auch die Berechnung des

Linear-Kernel	$\kappa(x_i, x_j) = x_i \cdot x_j + \gamma$
Polynom-Kernel	$\kappa(x_i, x_j) = (x_i \cdot x_j + \gamma)^{\delta}$
Gauss-/RBF-Kernel	$\kappa(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$
Sigmoid-Kernel	$\kappa(x_i, x_j) = \tanh(\alpha(x_i \cdot x_j) + \theta)$

Tabelle 1: Beispiele für häufig eingesetzte Kernelfunktionen im Maschinellen Lernen [HSS08].

Skalarproduktes im höherdimensionalen Raum einen asymptotischen Aufwand von $\mathcal{O}(D)$ hätte. An dieser Stelle kommt nun der in [BGV92] vorgestellte *Kernel-Trick* zum Tragen. Durch den Einsatz von sogenannten *Kernelfunktionen* lässt sich dieser Mehraufwand umgehen. Dabei handelt es sich um positiv definite Abbildungen $\kappa : X \times X \to \mathbb{R}$, die bei Eingabe von zwei Punkten aus dem Ursprungsraum X das Skalarprodukt der in den höherdimensionalen Raum abgebildeten Punkte berechnen:

$$\kappa(x, y) = \langle \phi(x), \phi(y) \rangle$$

Der Kernel-Trick besteht also darin, dass sich die Distanzberechnung auf Skalarprodukte von Eingabepunkten reduzieren lässt, für die keine explizite Kenntnis der Abbildung ϕ oder der abgebildeten Punkte $\phi(p)$ nötig ist. Dies können wir umsetzen, indem wir insgesamt $\mathcal{O}(n^2)$ Skalarprodukte berechnen oder direkt eine Kernelfunktion κ verwenden. Mit der Kernelfunktion κ können wir die Distanzberechnung aus 2.1 weiter vereinfachen:

$$\|\phi(p) - c\|^2 = \kappa(p, p) - \frac{1}{|S_c|} \sum_{x \in S_c} \kappa(p, x) - \frac{1}{|S_c|} \sum_{x \in S_c} \kappa(x, p) + \frac{1}{|S_c|^2} \sum_{x, y \in S_c} \kappa(x, y) \tag{2.2}$$

Tabelle 1 bietet eine Übersicht über häufig eingesetzte Kernelfunktionen im Maschinellen Lernen [HSS08]. Insbesondere die Gauss-Kernelfunktion und die Polynom-Kernelfunktion werden auch bei Support-Vektor-Maschinen gerne verwendet.

Eine auf Kernelfunktionen basierende Variante von (Lloyds) Algorithmus 1 folgt mit 2.2 unmittelbar. Der Algorithmus wird üblicherweise kurz „Kernel-k-means-Algorithmus“ genannt. Der Vorfaktor für gewichtetes k-means hat auf die Berechnungsvorschriften keine weiteren Auswirkungen, Algorithmus 3 kann dementsprechend für gewichtetes k-means erweitert werden. Der Algorithmus lässt sich mit einer asymptotischen Laufzeit von $\mathcal{O}(n^2(\tau + d))$ implementieren [DGK04, DGK07],

wobei τ die Anzahl an Iterationen der äußeren Schleife ist.

Im nächsten Unterabschnitt führen wir Verfahren ein, die mit Techniken aus der

Algorithmus 3: Kernel-k-means

Eingabe: $: P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+, \kappa : X \times X \to \mathbb{R}$
Ausgabe: $: k$-means-Clustering von P

Wähle zufällig k Zentren $c_1^{(0)}, \ldots, c_k^{(0)}$ aus P oder $\mathbb{R}^d$
$t \leftarrow 0$
repeat
    Berechne $\|\phi(p) - c_i^{(t)}\|^2$ für alle $p \in P$ und $i \in \{1, \ldots, k\}$ mit 2.2
    foreach $p \in P$ **do**
        $c^*(p) \leftarrow \arg\min_i \left(\left\| \phi(p) - c_i^{(t)} \right\| \right)$
    $t \leftarrow t + 1$
    for $i \leftarrow 1$ **to** k **do**
        $S_i^{(t)} \leftarrow \{p \mid c^*(p) = i\}$
        $c_i^{(t)} \leftarrow \frac{1}{|S_i^{(t)}|} \sum_{p_j \in S_i^{(t)}} p_j$
until Konvergenz oder $t > t_{max}$

linearen Algebra das Graphclusteringproblem lösen und in Kombination mit dem hier vorgestellten Algorithmus eine robuste Grundlage für unsere Zwecke bilden.

2.3.2 Spektrale Clustering Methoden

Beim spektralen Clustering besteht die Eingabe aus einem Graphen, dessen Knoten geclustert werden sollen. Informationen über die „Ähnlichkeit“ von Objekten werden in Form eines Kantengewichts der Kante zwischen den jeweiligen Objekten realisiert. Wir werden im Weiteren insbesondere Eigenvektoren und zugehörige Eigenwerte von Matrizen betrachten und rufen uns daher kurz die Definition dieser in Erinnerung.

Definition 2.3.3 (Eigenvektor und Eigenwert). Sei A eine quadratische Matrix. Das *Eigenwertproblem* besteht darin, eine Zahl λ und einen dazugehörigen Vektor $\vec{x} \neq \vec{0}$ zu finden, für die gilt:

$$A\vec{x} = \lambda\vec{x}$$

Die reelle oder komplexe Zahl λ heißt *Eigenwert* von A, der Vektor $\vec{x}$ heißt Eigenvektor von A.

Wenn G der Graph ist, durch den die Eingabemenge repräsentiert wird, dann ist die Adjazenzmatrix Adj^G des Graphen der Ausgangspunkt der spektralen

Clusteranalyse. Bevor wir mit konkreten Verfahren der spektralen Clusteranalyse beginnen können, benötigen wir einige grundlegende Definitionen der (spektralen) Graphentheorie.

Definition 2.3.4 (Grad-Matrix). Sei $G = (V, E)$ ein Graph mit $|V| = n$ Knoten. Die *Grad-Matrix* von G ist die $n \times n$-Diagonalmatrix D, die folgendermaßen definiert ist:

$$D_{i,j} = \begin{cases} \deg(v_i) & \text{falls } i = j, \\ 0 & \text{sonst} \end{cases}$$

Dabei ist $\deg(v_i)$ die Anzahl an Kanten, die zum Knoten v_i inzident sind. Diese Anzahl wird auch *Grad* des Knotens v_i genannt.

Mit Hilfe der Grad-Matrix eines Graphen lässt sich die sogenannte *Laplace-Matrix* des Graphen berechnen, welche eine Matrix-Repräsentation von Graphen ist, die aufschlussreiche Informationen über die Struktur und Beschaffenheit des Graphen erlaubt. Sie ist wie folgt definiert:

Definition 2.3.5 (Laplace-Matrix [Lux07]). Sei $G = (V, E)$ ein Graph ohne Schleifen mit Grad-Matrix D und (gewichteter) Adjazenzmatrix Adj^G. Die (nicht-normalisierte) *Laplace-Matrix* L von G ist definiert als:

$$L = D - Adj^G$$

Die Einträge der Matrix sind dementsprechend folgendermaßen definiert:

$$L_{i,j} = \begin{cases} \deg(v_i) & \text{falls } i = j, \\ -1 & \text{falls } i \neq j \text{ und } v_i \text{ inzident zu } v_j \text{ ist}, \\ 0 & \text{sonst} \end{cases}$$

Es existieren zwei normalisierte Formen der Laplace-Matrix, die *normalisierte* Laplace-Matrizen genannt werden. Sie sind wie folgt definiert:

$$\mathcal{L}_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$$
$$\mathcal{L}_{rw} = D^{-1} L$$

Die Eigenwerte der Laplace-Matrizen eines Graphen geben beispielsweise Aufschluss über die *Zusammenhangskomponenten* eines Graphen. Eine Zusammenhangskomponente definiert ist als Teilgraph, in dem jedes Paar von Knoten über einen Pfad miteinander verbunden ist. Das folgende Lemma illustriert die Zusammenhangsinformationen, die sich aus der Laplace-Matrix ablesen lassen:

Lemma 2.3.1 (Zusammenhang und Laplace-Matrix [Lux07]). Gegeben sei $G = (V, E)$ ein schleifenloser Graph mit nicht-negativen Gewichten und Laplace-Matrix L. Die Anzahl an Zusammenhangskomponenten von G ist gleich der Anzahl an Eigenwerten von L, die gleich Null sind.

Da die Eigenwerte einer Matrix auch „Spektrum“ genannt werden und man bei den hier vorgestellten Methoden die Eigenwerte der Laplace-Matrix nutzt, wird diese Technik „spektrale Clusteranalyse“ genannt.

Die wichtigsten Algorithmen, die auf dem Spektrum der Laplace-Matrix der Eingabe basieren, wollen wir im Folgenden vorstellen. Wir beginnen mit den Algorithmen 4 und 5 , welche die normalisierten Laplace-Matrizen verwenden und in den jeweils zitierten Papieren vorgestellt wurden.

Algorithmus 4: Nicht-normalisiertes spektrales Clustering [Lux07]

Eingabe: : Gewichtete Adjazenzmatrix $Adj^G \in \mathbb{R}^{n \times n}, k \in \mathbb{N}^+$
Ausgabe: : k Cluster $C_1, \ldots, C_k$

1 Berechne die nicht-normalisierte Laplace-Matrix L des Graphen G mit Adj^G
2 Bestimme Eigenvektoren $u_1, \ldots, u_k$ zu den k kleinsten Eigenwerten von L
3 Sei $U \in \mathbb{R}^{n \times k}$ die Matrix mit $u_1, \ldots u_k$ als Spaltenvektoren
4 Für $i \leftarrow \{1, \ldots, n\}$ sei $y_i \in \mathbb{R}^k$ der Vektor der i-ten Reihe von U
5 Clustere die Punkte $(y_i)_{i \in \{1,\ldots,n\}}$ mit einem k-means-Algorithmus in $C_1, \ldots, C_k$

Algorithmus 5: Normalisiertes spektrales Clustering [SM00]

Eingabe: : Gewichtete Adjazenzmatrix $Adj^G \in \mathbb{R}^{n \times n}, k \in \mathbb{N}^+$
Ausgabe: : k Cluster $C_1, \ldots, C_k$

1 Berechne die normalisierte Laplace-Matrix $\mathcal{L}_{rw}$ des Graphen G mit Adj^G
2 Bestimme Eigenvektoren $u_1, \ldots, u_k$ zu den k kleinsten Eigenwerten von $\mathcal{L}_{rw}$
3 Sei $U \in \mathbb{R}^{n \times k}$ die Matrix mit $u_1, \ldots u_k$ als Spaltenvektoren
4 Für $i \leftarrow \{1, \ldots, n\}$ sei $y_i \in \mathbb{R}^k$ der Vektor der i-ten Reihe von U
5 Clustere die Punkte $(y_i)_{i \in \{1,\ldots,n\}}$ mit einem k-means-Algorithmus in $C_1, \ldots, C_k$

Der in [SM00] vorgestellte Algorithmus sieht in seiner ursprünglichen Form zunächst eine Bipartitionierung anhand des Eigenvektors mit dem zweitkleinsten Eigenwert vor (dies entspricht einer Approximation des *sparsest cut*), die anschließend gegebenenfalls weiter partitioniert wird. Dieses Vorgehen ist jedoch analog zu

dem hier vor uns angegebenen Algorithmus.

Algorithmus 6: Normalisiertes spektrales Clustering [NJW01]

Eingabe: : Gewichtete Adjazenzmatrix $Adj^G \in \mathbb{R}^{n \times n}, k \in \mathbb{N}^+$

Ausgabe: : k Cluster $C_1, \ldots, C_k$

1 Berechne die normalisierte Laplace-Matrix $\mathcal{L}_{sym}$ des Graphen G mit Adj^G

2 Bestimme Eigenvektoren $u_1, \ldots, u_k$ zu den k kleinsten Eigenwerten von $\mathcal{L}_{sym}$

3 Sei $U \in \mathbb{R}^{n \times k}$ die Matrix mit $u_1, \ldots u_k$ als Spaltenvektoren

4 Sei $T \in \mathbb{R}^{n \times k}$ die Matrix mit den Einträgen $T_{i,j} = \frac{U_{i,j}}{\left(\sum_k U_{i,k}^2\right)^{\frac{1}{2}}}$

5 Für $i \leftarrow \{1, \ldots, n\}$ sei $y_i \in \mathbb{R}^k$ der Vektor der i-ten Reihe von T

6 Clustere die Punkte $(y_i)_{i \in \{1,\ldots,n\}}$ mit einem k-means-Algorithmus in $C_1, \ldots, C_k$

Der Algorithmus aus [NJW01] ist dem von [SM00] sehr ähnlich, verwendet jedoch die normalisierte Laplace-Matrix $\mathcal{L}_{sym}$. Außerdem ist für diesen Algorithmus eine zusätzliche Normalisierung nötig, deren Details in [Lux07] erläutert werden.

Das Berechnen aller Eigenvektoren benötigt asymptotischen Rechenaufwand von $\mathcal{O}(n^3)$, was für viele Anwendungen deutlich zu viel Rechenzeit ist. Beschränkt man sich auf die Berechnung von wenigen oder sogar nur einem einzelnen Eigenvektor, lässt sich die Komplexität signifikant verringern, beispielsweise unter Einsatz von Lanczos Algorithmus [Lan50]. Da für viele spektrale Clusteringalgorithmen die Berechnung der Eigenvektoren jedoch nach wie vor ein großer Flaschenhals ist, betrachten wir im nächsten Kapitel einen Ansatz, mit dem die Berechnung von Eigenvektoren für Graphclusterings prinzipiell vermieden werden kann.

3 Kernel k-means Methoden für spektrales Graphclustering

In diesem Kapitel wollen wir Verfahren zur spektralen Clusteranalyse von Graphen vorstellen und verbessern. Als Grundlage wollen wir den Kernel-k-means Algorithmus verwenden. Lloyds Algorithmus ist nach wie vor eine Heuristik, die in der Praxis mit geringen Ausführungszeiten Clusterings von akzeptabler Qualität berechnet. Unter Einsatz der Kernel Methode können wir die Qualität der berechneten Clusterings noch einmal steigern, ohne dabei signifikante Performanzeinbußen in Kauf nehmen zu müssen. Der Algorithmus ist zudem leicht zu implementieren. Gerade im Bereich der Clusteranalyse sollte dies nicht unterschätzt werden, da eine praktische Evaluation der Algorithmen in diesem anwendungsreichen Gebiet unerlässlich ist.

Das Kapitel ist folgendermaßen gegliedert: in Abschnitt 3.1 diskutieren wir, wie wir mit dem Kernel-k-means Algorithmus Graphclusteringprobleme (und umgekehrt) lösen können. In Abschnitt 3.2 zeigen wir, wie sich die Kernel Methode auch auf den Algorithmus k-means++ anwenden lässt und wie wir damit den Kernel-k-means Algorithmus für Graphclustering verbessern können.

3.1 Graphschnitte und Graphclustering mit gewichtetem Kernel k-means

Um die Graphclusteringprobleme aus Definition 2.2.3 mit dem Kernel-k-means Algorithmus zu lösen, müssen wir uns zunächst überlegen, wie wir diese in eine Eingabe für den Algorithmus transformieren können. Dass eine solche Transformation (bidirektional) möglich ist, haben bereits Dhillon, Guan und Kulis gezeigt [DGK04, DGK07]. Wir fassen daher zunächst den für uns relevanten Teil ihrer Ergebnisse zusammen.

Die grundlegende Idee besteht darin, zu zeigen, dass sowohl die Zielfunktion für Kernel-k-means als auch die (diversen) Zielfunktionen für Graphpartitionierungsprobleme als *Spurmaximimerungsprobleme* formuliert werden können. Die *Spur* (engl. *trace*) einer quadratischen $n \times n$-Matrix M ist dabei die Summe der Elemente auf der Hauptdiagonalen von M:

$$\operatorname{trace}(M) = \sum_{i=1}^{n} M_{i,i}$$

Weiterhin gilt für zwei quadratische Matrizen M, M':

1. $\text{trace}(MM') = \text{trace}(M'M)$
2. $\text{trace}(M^T M) = \|M\|_F^2$
3. $\text{trace}(M + M') = \text{trace}(M) + \text{trace}(M')$

Wir formen zunächst die Kernel-k-means Zielfunktion um und widmen uns dann den Graphpartitionierungsproblemen.

Gewichtetes Kernel-k-means. Für die Umformung des k-means-Problems legen wir das *gewichtete* Kernel-k-means Problem (engl. *weighted kernel k-means*, kurz WKKM) zu Grunde. Dieses lautet analog zu Definition 2.3.2 wie folgt. Unsere Eingabe besteht aus der gewichteten Punktmenge als $n \times d$-Matrix A, einer Gewichtsfunktion w, die einen Punkt-Zeilenvektor $\mathbf{p}$ aus A auf ein reelles Gewicht abbildet, und der Clusteranzahl k. Die Zielfunktion besteht wie gewohnt darin, die quadrierten euklidischen Distanzen der Punkte eines Clusters zu ihrem Clusterzentrum zu minimieren:

$$\min D(\{S_i\}_{i=1}^k) = \sum_{i=1}^{k} \sum_{\mathbf{p} \in S_i} w(\mathbf{p}) \|\phi(\mathbf{p}) - \mathbf{c}_i\|^2 \tag{3.1}$$

Dabei sind die $S_1, \ldots, S_k$ die Cluster, wobei ein Cluster S_i das Zentrum

$$\mathbf{c}_i = \frac{\sum_{\mathbf{p} \in S_i} w(\mathbf{p}) \phi(\mathbf{p})}{\sum_{\mathbf{p} \in S_i} w(\mathbf{p})}$$

hat. Wir setzen auch hier wieder den Kernel-Trick für die Distanzberechnungen ein. Dazu berechnen wir initial n^2 viele Skalarprodukte $\langle \phi(\mathbf{p}_i), \phi(\mathbf{p}_j) \rangle$ und speichern diese in einer $n \times n$-Kernelmatrix K mit $K_{i,j} = \kappa(\mathbf{p}_i, \mathbf{p}_j) = \langle \phi(\mathbf{p}_i), \phi(\mathbf{p}_j) \rangle$. Die Distanzberechnung von einem Punkt zu seinem Clusterzentrum können wir ebenfalls analog zu 2.2 mit Hilfe der Kernelmatrix umformulieren. Sie unterscheidet sich von dieser nur darin, dass die Punkte gewichtet sind:

$$\|\phi(\mathbf{p}_j) - \mathbf{c}_i\|^2 = K_{i,i} - \frac{2 \sum_{\mathbf{p}_j \in S_i} w(\mathbf{p}_j) K_{i,j}}{\sum_{\mathbf{p}_j \in S_i} w(\mathbf{p}_j)} + \frac{\sum_{\mathbf{p}_j, \mathbf{p}_l \in S_i} w(\mathbf{p}_j) w(\mathbf{p}_l) K_{j,l}}{\left(\sum_{\mathbf{p}_j \in S_i} w(\mathbf{p}_j) \right)^2} \tag{3.2}$$

Die Erweiterung von Algorithmus 3 zu Algorithmus 7 ergibt sich ebenfalls unmittelbar.

Algorithmus 7: Weighted Kernel-k-means

Eingabe: : Kernel-Matrix K, $k \in \mathbb{N}^+$, Punktgewichte w
Ausgabe: : k-means-Clustering der Eingabepunktmenge

1 Wähle k initiale Cluster $S_1^{(0)}, \ldots, S_k^{(0)}$; $t \leftarrow 0$
2 **foreach p do**
3 Bestimme $c^*(\mathbf{p}) \leftarrow \arg\min_{i=1}^{k} \|\phi(\mathbf{p}) - \mathbf{c_i}\|^2$ mit 3.2
4 **for** $i \leftarrow 1$ **to** k **do**
5 $S_i^{(t+1)} \leftarrow \{\mathbf{p} : c^*(\mathbf{p}) = i\}$
6 $t \leftarrow t + 1$
7 **if** $t < t_{max}$ ***and*** *nicht konvergiert* **then**
8 **go to** 2
9 **else**
10 **return** $S_1^{(t)}, \ldots, S_k^{(t)}$

Gewichtetes Kernel-k-means als Spurmaximierung. Wir formen als nächstes die Zielfunktion des gewichteten Kernel-k-means Problems in ein Spurmaximierungsproblem um. Dazu gehen wir zunächst davon aus, dass die Punktgewichte in einer $n \times n$-Diagonalmatrix W gespeichert sind, sodass $W_{l,l} = w(\mathbf{p}_j)$ für alle Punkte $\mathbf{p}_j$ und die Punktgewichte des Clusters S_i in der Diagonalmatrix W^i mit $W^j_{m,m} = w(\mathbf{p}_m)$ für alle Punkte $\mathbf{p}_m \in S_i$ gespeichert sind.

Sei nun s_i die Summe der Gewichte der Punkte in Cluster i, also

$$s_i = \sum_{\mathbf{p}_j \in S_i} w(\mathbf{p}_j) = \sum_{\mathbf{p}_j \in S_i} W^i_{j,j}$$

Wir betrachten die $n \times k$-Matrix Z mit den Einträgen

$$Z_{j,i} = \begin{cases} \frac{1}{\sqrt{s_i}}, & \text{falls } \mathbf{p}_j \in S_i, \\ 0 & \text{sonst} \end{cases}$$

sowie die (nicht explizit berechnete) Matrix $\Phi_i = [\phi(\mathbf{p}_1), \ldots, \phi(\mathbf{p}_m)]$ von abgebildeten Punkten des Clusters $S_i = \{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$. Mit diesen können wir die Zielfunktion des gewichteten Kernel-k-means Problems (3.1) umformen. Wir erinnern uns, dass diese die Minimierung der Summe der Intraclusterabstände ist. Notieren wir den Intraclusterabstand eines Clusters S_i mit $d(S_i) = \sum_{\mathbf{p} \in S_i} w(\mathbf{p}) \|\phi(\mathbf{p}) - \mathbf{c}_i\|^2$, können wir für den Zielfunktionswert $D(\{S_i\}_{i=1}^k)$ verkürzt schreiben:

$$D(\{S_i\}_{i=1}^k) = \sum_{i=1}^{k} d(S_i)$$

Der Zentroidvektor c_i des Clusters S_i berechnet sich in unserer ursprünglichen Problemformulierung durch

$$\mathbf{c}_i = \frac{\sum_{\mathbf{p}\in S_i} w(\mathbf{p})\phi(\mathbf{p})}{\sum_{\mathbf{p}\in S_i} w(\mathbf{p})}$$

Diese Berechnungsvorschrift ist äquivalent zu

$$\mathbf{c}_i = \Phi_i \frac{W_j \mathbf{e}}{s_j},$$

wobei $\mathbf{e}$ der Einheitsvektor der passenden Dimension ist. Wir können alternativ unter Einsatz der Matrix Z auch direkt eine Matrix C aller Clusterzentren bestimmen, wenn wir für die gesamte Eingabepunktmenge die (ebenfalls nicht explizit berechnete) Abbildungsmatrix $\Phi = [\Phi_1, \ldots, \Phi_k] = [\phi(\mathbf{p}_1), \ldots, \phi(\mathbf{p}_n)]$ verwenden:

$$C = \Phi W Z Z^T$$

In diesem Fall entspricht das Zentrum eines einzelnen Clusters S_i

$$\mathbf{c}_i = \left(\Phi W Z Z^T\right)_{\cdot i}$$

wobei wir für eine Matrix M mit $M_{\cdot i}$ die i-te Spalte von M meinen. Damit ist es uns nun möglich, den Zielfunktionswert so zu formulieren, dass wir später eine Äquivalenz zur Graphpartitionierung sehen werden. Mit $Y = W^{\frac{1}{2}} Z$ gilt:

$$\begin{aligned}
D(\{S_i\}_{i=1}^k) &= \sum_{i=1}^{k} \sum_{\mathbf{p}\in S_i} w(\mathbf{p}) \|\phi(\mathbf{p}) - \mathbf{c}_i\|^2 \\
&= \sum_{j=1}^{n} W_{j,j} \left\| \Phi_{\cdot j} - \left(\Phi W Z Z^T\right)_{\cdot j} \right\|^2 \\
&= \sum_{j=1}^{n} W_{j,j} \left\| \Phi_{\cdot j} - \left(\Phi W^{\frac{1}{2}} Y Y^T W^{-\frac{1}{2}}\right)_{\cdot j} \right\|^2 \\
&= \sum_{j=1}^{n} \left\| \Phi_{\cdot j} \left(W_{j,j}\right)^{\frac{1}{2}} - \left(\Phi W^{\frac{1}{2}} Y Y^T\right)_{\cdot j} \right\|^2 \\
&= \left\| \Phi W^{\frac{1}{2}} - \Phi W^{\frac{1}{2}} Y Y^T \right\|_F^2
\end{aligned}$$

Der ursprüngliche Zielfunktionswert ist damit auf die (quadrierte) Frobenius-Norm von $\Phi W^{\frac{1}{2}} - \Phi W^{\frac{1}{2}} Y Y^T$ transformiert. Wir können nun die oben genannte Verbindung zwischen der Frobenius-Norm und der Matrix-Spur ausnutzen, um zur

beabsichtigten Spur-Maximierung zu gelangen. Wir erinnern uns, dass für eine Matrix M gilt, dass $\text{trace}(M^T M) = \|M\|_F^2$. Wir können somit weiter umformen:

$$\begin{aligned} D(\{S_i\}_{i=1}^k) &= \text{trace}(W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}} - W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}}YY^T \\ &\quad - YY^T W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}} + YY^T W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}}YY^T) \\ &= \text{trace}(W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}}) - \text{trace}(Y^T W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}}Y) \end{aligned}$$

In der Kernel-Matrix K speichern wir sämtliche Skalarprodukte der projizierten Punkte, demnach ist per Definition $K = \Phi^T\Phi$. Die Kernel-Matrix ist genau wie die Gewichts-Matrix W konstant, da beide Teil der Eingabe sind. Dementsprechend ist auch $\text{trace}(W^{\frac{1}{2}}KW^{\frac{1}{2}})$ eine Konstante. Die Minimierungszielfunktion 3.1 ist daher gleichzusetzen mit der Maximierungszielfunktion des folgenden Spurmaximierungsproblems:

$$\max_Y \text{trace}(Y^T W^{\frac{1}{2}}KW^{\frac{1}{2}}Y) \tag{3.3}$$

Da das eigentliche Clustering von Algorithmus 7 offenbar einer spurmaximierenden Wahl der Matrix Y gleichkommt, wollen wir abschließend noch kurz einen genaueren Blick auf diese werden. Die Matrix $Y = W^{\frac{1}{2}}Z$ ist eine orthonormale Diagonalmatrix, das heißt $Y^T Y = I$. Ihre Einträge sehen dementsprechend wie folgt aus:

$$Y = \begin{pmatrix} \frac{W_1^{\frac{1}{2}}\mathbf{e}}{\sqrt{s_1}} & & & \\ & \frac{W_2^{\frac{1}{2}}\mathbf{e}}{\sqrt{s_2}} & & \\ & & \cdots & \\ & & & \frac{W_k^{\frac{1}{2}}\mathbf{e}}{\sqrt{s_k}} \end{pmatrix}$$

Graphpartitionierung als Spurmaximierung. Es ist möglich, für alle in Definition 2.2.3 vorgestellten Problemstellungen ein äquivalentes Spurmaximierungsproblem anzugeben. Wir beschränken uns hier zunächst auf die beiden Zielfunktionen mit praktischer Relevanz für das Evaluations-Kapitel dieser Arbeit, nämlich die Ratio Association und den Normalized Cut. Wir beginnen mit der Ratio Association.

Wir führen zunächst einen Indikator-Vektor $\mathbf{x}_c$ der Dimension n für jede Partition beziehungsweise jedes Cluster c ein, sodass $\mathbf{x}_c(i) = 1$ genau dann, wenn das Cluster c den Knoten i enthält. Die Zielfunktion bei der Ratio Association lautet

$$\max \sum_{c=1}^{k} \frac{w(V_c, V_c)}{|V_c|}$$

Wir nehmen an, dass A die (gewichtete) Adjazenzmatrix des Eingabegraphen ist und definieren zusätzlich

$$\tilde{\mathbf{x}}_c = \frac{\mathbf{x}_c}{(\mathbf{x}_c^T \mathbf{x}_c)^{\frac{1}{2}}}$$

Das Produkt $\mathbf{x}_c^T \mathbf{x}_c$ entspricht der Größe der Partition c:

$$|V_c| = \mathbf{x}_c^T \mathbf{x}_c$$

Die Summe der Kantengewichte der Partition $w(V_c, V_c)$ können wir über folgendes Produkt abbilden:

$$w(V_c, V_c) = \mathbf{x}_c^T A \mathbf{x}_c$$

Damit können wir eine erste Transformation der Zielfunktion vornehmen:

$$\max \sum_{c=1}^{k} \frac{w(V_c, V_c)}{|V_c|} \Leftrightarrow \max \sum_{c=1}^{k} \frac{\mathbf{x}_c^T A \mathbf{x}_c}{\mathbf{x}_c^T \mathbf{x}_c} \Leftrightarrow \max \sum_{c=1}^{k} \tilde{\mathbf{x}}_c^T A \tilde{\mathbf{x}}_c$$

Betrachten wir $\tilde{X}$ als Matrix deren c-te Spalte der Vektor $\tilde{\mathbf{x}}_c$ ist, können wir die Zielfunktion umformen zu

$$\max_{\tilde{X}} \operatorname{trace}(\tilde{X}^T A \tilde{X}) \tag{3.4}$$

Wenn wir in 3.3 für die Gewichtsmatrix W die Einheitsmatrix wählen, also $W = \mathbb{1}_n$ und die Adjazenzmatrix als Kernel-Matrix setzen, sodass $K = A$, dann sind 3.4 und 3.3 äquivalent. Auf diese Weise können wir das Ratio Association Problem mit unserem Algorithmus für gewichtetest Kernel-k-means lösen.

Für den Normalized Cut gehen wir ähnlich vor. Unsere ursprüngliche Zielfunktion lautet

$$\min \sum_{c=1}^{k} \frac{w(V_c, V \setminus V_c)}{w(V_c, V)}$$

Zunächst wollen wir die Funktion in ein Maximierungsproblem umwandeln. Wir beobachten, dass die Zielfunktion äquivalent ist zu

$$\max \sum_{c=1}^{k} \frac{w(V_c, V_c)}{w(V_c, V)}$$

Wir benötigen zur Umformung des Normalized Cut die diagonale Gradmatrix D des Graphen, der der Adjazenzmatrix A zu Grunde liegt. Wir verwenden wieder

die Indikator-Vektoren und definieren zusätzlich

$$\hat{\mathbf{x}}_c = \frac{\mathbf{x}_c}{(\mathbf{x}_c^T D \mathbf{x}_c)^{\frac{1}{2}}}$$

Unter Verwendung der Gradmatrix können wir die Variante des Normalized Cut als Maximierungsproblem wie folgt umformen:

$$\max \sum_{c=1}^{k} \frac{w(V_c, V_c)}{w(V_c, V)} \Leftrightarrow \max \sum_{c=1}^{k} \frac{\mathbf{x}_c^T A \mathbf{x}_c}{\mathbf{x}_c^T D \mathbf{x}_c} \Leftrightarrow \max \sum_{c=1}^{k} \hat{\mathbf{x}}_c^T A \hat{\mathbf{x}}_c$$

Wir nehmen analog zur Ratio Association an, dass die Matrix $\hat{X}$ die Spaltenmatrix der Vektoren $\hat{\mathbf{x}}_c$ ist. Mit $\tilde{Y} = D^{\frac{1}{2}} \hat{X}$ lautet das Spurmaximierungsproblem für den Normalized Cut folgendermaßen:

$$\max_{\tilde{Y}} \operatorname{trace}(\tilde{Y}^T D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tilde{Y}) \tag{3.5}$$

Das Problem in 3.5 ist äquivalent zu 3.3, wenn wir als Gewichtsmatrix $W = D$ und als Kernel-Matrix $K = D^{-1} A D^{-1}$ setzen.

Wir wollen abschließend noch anmerken, dass die Adjazenzmatrix A positiv definit sein muss, damit sie als Kernel-Matrix in Algorithmus 7 garantiert zu einer iterativen Verringerung des gewichteten Kernel-k-means Zielfunktionswertes führt. Um dies in der Praxis sicherzustellen ist bei der Wahl der Kernel-Matrix gegebenenfalls das Aufaddieren einer „Verschiebung“ auf die ursprüngliche Kernel-Matrix nötig. Wir gehen auf dieses Detail im Experimente-Kapitel noch genauer ein.

3.2 Die Kernel Methode für k-means++

Ein wichtiges Merkmal von Algorithmus 7 ist, dass er grundsätzlich ohne Methoden der spektralen Clusteranalyse auskommt. Wir müssen uns jedoch wie auch bei Lloyds Algorithmus ohne Kernel Methode mit der geeigneten initialen Wahl der Clusterzentren beschäftigen. Zunächst einmal wählen wir diese wie schon zuvor zufällig aus der Eingabepunktmenge. Diese Vorgehensweise ist durchaus praktikabel, die experimentelle Evaluation aus [DGK04, DGK07] zeigt jedoch, dass sich mit einer geschickteren Wahl die Qualität des letztlich berechneten Clusterings noch signifikant steigern lässt. Die Autoren verwenden für die Initialisierung die spektrale Clusteranalyse, was wiederum den Aufwand der Eigenvektor-Berechnung mit sich bringt.

Konkret wird mit einem Verfahren aus [GL96] eine Problemrelaxion von 3.3/ 3.4 beziehungsweise von 3.3/ 3.5 gelöst. Relaxieren wir die Wahl von Y beziehungsweise $\tilde{Y}$ auf eine beliebige orthonormale Matrix, dann sind diese von der Form $Y = V_k Q$ beziehungsweise $\tilde{Y} = \tilde{V}_k Q$. Dabei ist Q eine beliebige orthogonale $k \times k$-Matrix, V_k die Matrix der k größten Eigenvektoren von $W^{\frac{1}{2}} K W^{\frac{1}{2}}$ und $\tilde{V}_k$ die Matrix der k größten Eigenvektoren von $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$.

Die Berechnung der jeweiligen k größten Eigenvektoren wird mit wachsender Clusteranzahl k aufwändig, wie wir schon in Abschnitt 2.3 beobachtet haben. Intuitiv ist daher die Fragestellung, ob es möglich ist, eine Initialisierung ohne Eigenvektor-Berechnungen vorzunehmen, die dennoch deutlich verbesserte Clusterings mit dem Kernel-k-means-Algorithmus ermöglicht.

In Abschnitt 2.1 haben wir bereits den Algorithmus k-means++ von Arthur und Vassilvitskii [AV07] kennengelernt. Die initiale Wahl der Cluster erfolgt hier gemäß einer Wahrscheinlichkeitsverteilung der Punkte als mögliche Clusterzentren, die proportional zur quadrierten euklidischen Distanz zum jeweils nächstgelegenen bereits gewählten Zentrum ist. Wir zeigen nun, dass die Kernel Methode auch auf den Algorithmus k-means++ angewendet werden kann. Damit ist es uns möglich, Algorithmus 3 (und damit auch Algorithmus 7) in eine Kernel-basierte Variante von k-means++ umzuwandeln, die wir Kernel-k-means++ nennen.

Wir erinnern uns, dass wir im Zusammenhang mit k-means++ für einen Punkt x aus der Eingabe-Punktmenge P mit $D(x)$ die geringste Distanz von x zum nächstgelegenen bereits gewählten Zentrum notieren. Der Punkt x wird als nächstes Zentrum mit der Wahrscheinlichkeit

$$\frac{D(x)^2}{\sum_{y \in P} D(y)^2}$$

gewählt. Wir können demnach die Kernel Methode auf den Algorithmus k-means++ anwenden, wenn wir für jeden Eingabepunkt x den Wert $D(\phi(x))$ beziehungsweise $D(\phi(x))^2$ berechnen können. Dazu zeigen wir nun, dass wir diese Werte wie schon zuvor beim Kernel-k-means-Algorithmus ausschließlich über die Kernelfunktion κ ohne explizite Kenntnis der abgebildeten Punkte oder der Abbildung ϕ selbst berechnen können.

Wir nehmen zunächst an, dass die Zentren $c_1, \ldots, c_l$ mit $l \in \{1, \ldots, k-1\}$ bereits gewählt worden sind. Dann können wir $D(x)$ formal notieren als das Minimum über alle quadrierten euklidischen Distanzen von x zu jedem der bisher gewählten Zentren c_i:

$$D(x) = \min_{i=1 \ldots l} \|x - c_i\|^2$$

Wenden wir die Kernel Methode an, berechnet sich $D(x)$ dementsprechend folgendermaßen:

$$D(x) = \min_{i=1\dots l} \|\phi(x) - \phi(c_i)\|^2$$

Demnach genügt es zu zeigen, dass wir $\|\phi(x) - \phi(c_i)\|^2$ ausschließlich mit der Kernelfunktion berechnen können. Wir haben bereits gesehen, dass die Kernelfunktion gerade dafür gedacht ist. Es gilt:

$$\|\phi(x) - \phi(c_i)\|^2 = \kappa(x,x) - 2\kappa(x,c_i) + \kappa(c_i,c_i) \tag{3.6}$$

Mit den selben Überlegungen können wir auch zeigen, dass sich der Wert von $D(x)^2$ mit der Kernel Methode berechnen lässt. Zunächst lässt sich auch $D(x)^2$ als Minimum formulieren:

$$D(x)^2 = \left(\min_{i=1\dots l} \|\phi(x) - \phi(c_i)\|^2\right)^2$$

Dazu müssen wir effektiv $\left(\|\phi(x) - \phi(c_i)\|^2\right)^2$ berechnen. Auch dies lässt sich mit der Kernelfunktion umsetzen:

$$\begin{aligned}\left(\|\phi(x) - \phi(c_i)\|^2\right)^2 &= \left(\kappa(x,x) - 2\kappa(x,c_i) + \kappa(c_i,c_i)\right)^2 \\ &= \left(\kappa(x,x)\right)^2 - 4\kappa(x,x)\kappa(x,c_i) + 2\kappa(x,x)\kappa(c_i,c_i) \\ &\quad + 4\kappa(x,c_i)\kappa(x,c_i) - 4\kappa(x,c_i)\kappa(c_i,c_i) + \left(\kappa(c_i,c_i)\right)^2 \end{aligned} \tag{3.7}$$

Der Algorithmus Kernel-k-means++ wird damit die logische Weiterentwicklung von k-means++ auf Basis der Kernel Methode.

Algorithmus 8: Kernel-k-means++

Eingabe: : $P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+$, Kernelfunktion κ
Ausgabe: : k initiale Clusterzentren für P

1 Wähle c_1 zufällig gleichverteilt aus P
2 **for** $i \leftarrow 2$ **to** k **do**
3 Berechne $D(x')^2$ mit 3.7 für alle $x' \in P \setminus \{c_1, \dots, c_{i-1}\}$
4 Wähle den Punkt $x' \in P \setminus \{c_1, \dots, c_{i-1}\}$ als Zentrum c_i mit Wahrscheinlichkeit $\frac{D(x')^2}{\sum_{x \in P} D(x)^2}$
5 Führe den Kernel-k-means-Algorithmus mit den initialen Clusterzentren $c_1, \dots, c_k$ aus.

Wir zeigen nun, dass auch Kernel-k-means++ eine $\mathcal{O}(\log k)$-Approximation für das k-means-Problem ist.

Literatur

[ADHP09] Aloise, Daniel ; Deshpande, Amit ; Hansen, Pierre ; Popat, Preyas: NP-hardness of Euclidean sum-of-squares clustering. In: *Machine Learning* (2009), S. 245–248

[AV07] Arthur, David ; Vassilvitskii, Sergei: k-means++: The advantages of careful seeding. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, 2007, S. 1027–1035

[BGV92] Boser, Bernhard E. ; Guyon, Isabelle ; Vapnik, Vladimir: A Training Algorithm for Optimal Margin Classifiers. In: *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992.*, 1992, S. 144–152

[CKV13] Celebi, M. E. ; Kingravi, Hassan A. ; Vela, Patricio A.: A comparative study of efficient initialization methods for the k-means clustering algorithm. In: *Expert Syst. Appl.* 40 (2013), Nr. 1, S. 200–210

[Cov65] Cover, T.M.: Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. In: *Electronic Computers, IEEE Transactions on* EC-14 (1965), June, Nr. 3, S. 326–334

[DGK04] Dhillon, Inderjit ; Guan, Yuqiang ; Kulis, Brian: A Unified View of Kernel k-means, Spectral Clustering and Graph Cuts / University of Texas at Austin. 2004 (TR-04-25). – Forschungsbericht

[DGK07] Dhillon, Inderjit S. ; Guan, Yuqiang ; Kulis, Brian: Weighted Graph Cuts without Eigenvectors A Multilevel Approach. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 29 (2007), Nr. 11, S. 1944–1957

[GL96] Golub, Gene H. ; Loan, Charles F.: *Matrix computations*. 3. Auflage. Johns Hopkins University Press, 1996

[HSS08] Hofmann, Thomas ; Schölkopf, Bernhard ; Smola, Alexander J.: Kernel methods in machine learning. In: *Ann. Statist.* 36 (2008), 06, Nr. 3, S. 1171–1220

[KL70] Kernighan, B. W. ; Lin, S.: An Efficient Heuristic Procedure for Partitioning Graphs. In: *The Bell System Technical Journal* 49 (1970), Nr. 1, S. 291–307

[Lan50] Lanczos, Cornelius: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. In: *Journal of Research of the National Bureau of Standards* 45 (1950), S. 255–282

[Llo82] Lloyd, Stuart P.: Least squares quantization in PCM. In: *IEEE Transactions on Information Theory* 28 (1982), Nr. 2, S. 129–136

[Lux07] Luxburg, Ulrike von: A Tutorial on Spectral Clustering. In: *Statistics and Computing* 17 (2007), Nr. 4, S. 395–416

[MS84] Megiddo, Nimrod ; Supowit, Kenneth J.: On the Complexity of Some Common Geometric Location Problems. In: *SIAM J. Comput.* 13 (1984), Nr. 1, S. 182–196

[NJW01] Ng, Andrew Y. ; Jordan, Michael I. ; Weiss, Yair: On Spectral Clustering: Analysis and an algorithm. In: *Advances In Neural Information Processing Systems*, 2001, S. 849–856

[ORSS12] Ostrovsky, Rafail ; Rabani, Yuval ; Schulman, Leonard J. ; Swamy, Chaitanya: The effectiveness of lloyd-type methods for the k-means problem. In: *J. ACM* 59 (2012), Nr. 6, S. 28

[Sch14] Schmidt, Melanie: *Coresets and Streaming Algorithms for the k-means Problem and Related Clustering Objectives*, Technische Universität Dortmund, Diss., 2014

[SM00] Shi, Jianbo ; Malik, Jitendra: Normalized Cuts and Image Segmentation. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 22 (2000), Nr. 8, S. 888–905

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 4. Februar 2015

Lukas Pradel