technische universität
dortmund

Masterarbeit

Kernel k-means Methoden zur spektralen Clusteranalyse von Graphen

Lukas Pradel
2. Januar 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

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 die von der klassischen Clusteranalyse sehr unterschiedlichen 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 d-dimensionalen Punkten, welche der erste Teil der Eingabe ist, gemäß einer Cluster-Zielfunktion möglichst optimal in genau k Cluster unterteilt werden, wobei k der ganzzahlige zweite Teil der Eingabe 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, welche als Zielfunktion für die beiden bekanntesten Clustering-Problemstellungen dient.

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 oder sogar beliebige Punkte aus $\mathbb{R}^d$ als initiale Clusterzentren. 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 Zuweisung von Punkten zum nächstgelegenen Cluster und die Neuberechnung der neuen Zentren 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. Die Güte des berechneten Clusterings hängt maßgeblich von der initialen Wahl der Cluster ab. Der Algorithmus k-means++ [AV07] setzt genau an dieser Stelle an: er berechnet auf einfache, aber dennoch geschickte Art und Weise die initialen Cluster und führt anschließend mit diesen die übrigen Schritte von Lloyds Algorithmus durch. Der Algorithmus wählt zunächst ein einzelnes Clusterzentrum c_1 zufällig gleichverteilt aus der Eingabe-Punktmenge P und wählt alle weiteren Clusterzentren sukzessive nach der folgenden Vorschrift, 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 $\frac{D(x')^2}{\sum_{x \in P} D(x)}$ 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** $i \leftarrow 1$ **to** k **do**
3 Wähle den Punkt $x' \in P \setminus \{c_1, \ldots, c_{i-1}\}$ als Zentrum c_i mit Wahrscheinlichkeit $\frac{D(x')^2}{\sum_{x \in P} D(x)}$
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 eine konkrete Anwendung partitionierender Clusteringverfahren.

2.2 Graphen und Clusteranalyse von Graphen

Gerade im Umgang mit großen Datenmengen hat sich in der Informatik der *Graph* als geeignete Datenstruktur erwiesen. Er ist wie folgt definiert.

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 partitionierender Clusteringverfahren übertragen. 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 $V = S \cup T$. 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.

Beim Clustering von Punktmengen lagen für die Ähnlichkeitsrelation 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:

$$\max_{V_1,\ldots,V_k} \sum_{c=1}^{k} \frac{w(V_c, V_c)}{|V_c|}$$

2. **Ratio Cut**. Beim Ratio Cut wird der Schnitt zwischen jeweils einem Cluster und allen anderen Punkten im Graphen minimiert:

$$\min_{V_1,\ldots,V_k} \sum_{c=1}^{k} \frac{w(V_c, V \setminus V_c)}{|V_c|}$$

3. **Kernighan-Lin**. Bei dem in [KL70] vorgestellten Optimierungskriterium werden die Intra-Clusterabstände ähnlich der Ratio Association minimiert, allerdings müssen alle Partition hier zusätzlich die selbe Größe haben. Die hier vorgestellte Zielfunktion ist von 2 auf k Partitionen verallgemeinert:

$$\min_{V_1,\ldots,V_k} \sum_{c=1}^{k} \frac{w(V_c, V_c)}{|V_c|} \text{ s.t. } |V_c| = \frac{|V|}{k} \, \forall \, c \in \{1, \ldots, k\}$$

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.

$$\min_{V_1,\ldots,V_k} \sum_{c=1}^{k} \frac{w(V_c, V \setminus V_c)}{w(V_c, V)}$$

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 gemäß der k-means Zielfunktion 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 Ähnlichkeit. Üblicherweise wird dabei das *Skalarprodukt* eingesetzt. Dabei beziehen wir 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 $\mathbb{R}$-Vektorraum. 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 ein $\chi \in \mathcal{V}$ 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). 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:

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

Ein Clusterzentrum c_i ist dementsprechend definiert durch

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

wobei S_i die Menge aller dem Cluster 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):

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

Diese Beobachtung allein hilft uns noch nicht weiter, da auch die Berechnung des 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, eine Kernelfunktion zur Berechnung der Skalarprodukte im höherdimensionalen Raum zu verwenden, für die keine explizite Kenntnis der Abbildung ϕ oder der abgebildeten Punkte $\phi(p)$ nötig ist. Mit der

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.

Kernelfunktion κ können wir nun 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 die am häufigsten eingesetzten Kernelfunktionen. Insbesondere die Gauss-Kernelfunktion und die Polynom-Kernelfunktion werden auch bei Support-Vektor-Machines 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.

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 $t > t_{max}$

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 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

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

[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. – 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

[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

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

[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

[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 2. Januar 2015

Lukas Pradel