Graph-Kerne

Karsten Michael Borgwardt

University of Cambridge
kmb51@cam.ac.uk

Abstract: Von sozialen Online-Netzwerken wie `myspace` bis hin zu Protein-Interaktionsnetzwerken — Netzwerke spielen eine immer größere Rolle in unserer Gesellschaft, sowohl in der Wissenschaft als auch in der Wirtschaft. Aus verschiedensten Gründen möchten wir den Aufbau dieser Netzwerke verstehen, zum Beispiel um herauszufinden, welche Gene und Proteine am Entstehen von Krankheiten beteiligt sind, oder um jene Kunden zu erkennen, die eine Vielzahl anderer Kunden in ihrem Kaufverhalten beeinflussen.

Graphen sind die natürliche Datenstruktur, um diese Netzwerke darzustellen. Leider existieren jedoch nur wenige algorithmische Verfahren, um Daten in Form von Graphen zu analysieren, geschweige denn *effizient* und *statistisch-fundiert* zu vergleichen. Inhalt meiner Doktorarbeit *Graph Kernels* (Deutsch *Graph-Kerne*) war die Entwicklung effizienter und statistisch-fundierter Algorithmen zur Analyse von Graphen. Die Ergebnisse meiner Arbeit umfassen hoch-effiziente Ähnlichkeitsmaße auf Graphen, den ersten Zwei-Stichproben-Test auf Graphen und ein neuartiges Verfahren, um informative Subgraphen einer Gruppe von Graphen zu bestimmen. Mit diesen Algorithmen stehen vielen Wissenschaftszweigen nun zahlreiche neue Werkzeuge zur Verfügung, um Netzwerkdaten zu untersuchen.

1 Graphen

Graphen sind die Datenstruktur der Wahl, um Objekte und ihre Beziehungen darzustellen: Objekte werden in Form von *Knoten* modelliert und Beziehungen zwischen diesen Knoten in Form von *Kanten*.

In vielen Wissenschaftszweigen sind Graphen populärer denn je. Biologen und Bioinformatiker untersuchen zelluläre Netzwerke, zum Beispiel Protein-Interaktionsnetzwerke und Genregulationsnetzwerke, um das Zusammenspiel von Genen und Protein zu erkunden. Soziologen analysieren soziale Netzwerke, zum Beispiel von Online-Communities wie `myspace`, um deren Aufbau zu verstehen. Auch die Wirtschaft studiert soziale Netzwerke, zum Beispiel im Bereich Marketing, um Trendsetter zu finden, also Personen, die andere in ihrem Kaufverhalten beeinflussen.

Beim Analysieren dieser Graphen besteht die grundlegende algorithmische Aufgabe darin, die Ähnlichkeit zweier Graphen zu messen. Zwar widmet sich die Chemoinformatik schon seit Jahrzehnten der Suche nach ‘guten’ Ähnlichkeitsmaßen für Graphen [TC00], aber die Moleküldaten der Chemoinformatik unterscheiden sich von jenen der Bioinformatik und der sozialen Netzwerkanalyse: Während Moleküle meist Dutzende Knoten ent-

halten, bestehen Protein-Interaktionsnetzwerke aus Tausenden, Internet-Communities sogar aus Zehn- und Hunderttausenden von Knoten. Die klassischen Verfahren sind nicht effizient genug, um diese großen Graphen zu analysieren. Aus Anwendungssicht besteht daher ein dringender Bedarf an effizienten Algorithmen zur Analyse großer Graphen. Aus theoretischer Sicht ist es erstrebenswert, dass diese Effizienz nicht allein mittels Heuristiken erreicht wird, von denen wir nicht wissen, ob die von ihnen gefundene Lösung nah an der optimalen Lösung liegt.

Im Machine Learning ist ein reicher Schatz an solchen theoretisch-fundierten Algorithmen zur Datenanalyse vorhanden, die sogenannten *Kern-Verfahren*. Der erste Teil meiner Doktorarbeit bechäftigte sich damit, diese Algorithmen für die Analyse von großen Graphen nutzbar zu machen, durch das Definieren effizient berechenbarer Ähnlichkeitsmaße auf Graphen (siehe Abschnitt 3). Im zweiten Teil meiner Doktorarbeit entwickelte ich neue Kern-Verfahren zur Graphenanalyse, darunter den ersten Zwei-Stichproben-Test auf Graphen (siehe Abschnitt 4) und ein Feature-Selektionsverfahren auf Graphen (siehe Abschnitt 5).

2 Kern-Verfahren

Da Kerne (*kernels*) und Kern-Verfahren (*kernel methods*) in allen hier vorgestellten Verfahren eine tragende Rolle spielen, stellen wir diese im folgenden kurz vor. Die kurze anschauliche Erklärung ist, dass ein Kern ein Ähnlichkeitsmaß ist und ein Kern-Verfahren ein Algorithmus, der auf einem solchen Ähnlichkeitsmaß, einem Kern, basiert.

Kerne haben ihren Ursprung in der sogenannten Support Vector Machine [Vap95], einem Verfahren für binäre Klassifikationsprobleme. Die Idee der Support Vector Machines besteht darin, zwei Klassen A und B von Objekten durch eine Hyperebene (bzw. eine Gerade im zweidimensionalen Fall) zu separieren. Dabei soll der Rand (*Margin*) maximiert werden, der Abstand zwischen der Hyperebene und den nächstgelegenen Punkten aus A und B. Oft ist es schwierig bis unmöglich, eine Hyperebene zu finden, die A von B trennt (siehe Abbildung 1 links). In diesem Fall bedient man sich des sogenannten *Kern-Tricks*, der aus zwei Aspekten besteht:

Erstens sucht man die Hyperebene nicht im ursprünglichen Eingaberaum, sondern bildet A und B mittels einer Funktion ϕ in einen (meist höher-dimensionalen) Raum ab, den sogenannten Feature-Raum, und berechnet die Hyperebene dort. Bei geeigneter Wahl von ϕ ist man oft in der Lage, A und B deutlich besser durch eine Hyperebene zu trennen als im Eingaberaum (siehe Abbildung 1 rechts).

Zweitens kann die Hyperebene im Feature-Raum mittels inneren Produkten zwischen den Punkten im Feature-Raum bestimmt werden. In anderen Worten, auf die Datenpunkte $x \in A \cup B$ und $x' \in A \cup B$ wird im Feature-Raum nur in Form von Skalarprodukten $\langle \phi(x), \phi(x') \rangle$ zugegriffen. Um dieses Skalarprodukt zu berechnen, könnte man naiv erst $\phi(x)$ und $\phi(x')$ bestimmen und anschließend ihr inneres Produkt. Der Kern-Trick besteht darin, diese Berechnung in einem Schritt durchzuführen, indem man eine Kern-Funktion (*kernel function*, kurz *kernel* oder *Kern*) $k(x, x') = \langle \phi(x), \phi(x') \rangle$ definiert und evaluiert.

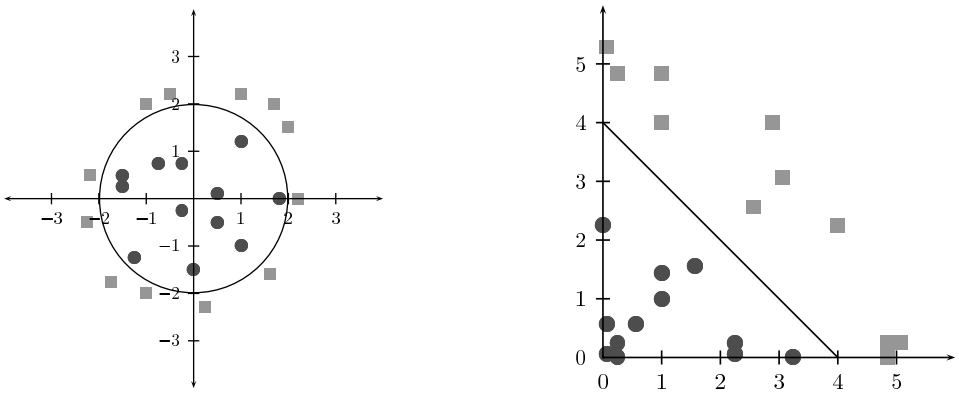

Abbildung 1: Der Kern-Trick. Wenn wir zwei Klassen von Punkten (Klasse A = Punkte, Klasse B = Rechtecke) nicht durch eine Hyperebene (schwarze Linie) im Eingaberaum (**links**) trennen können, bilden wir sie in einen Feature-Raum (**rechts**) ab, um bessere Separierbarkeit zu erreichen. Alle Berechnungen im Feature-Raum (**rechts**) können durch das Evaluieren einer Kern-Funktion auf den Objekten im Eingaberaum (**links**) erfolgen.

Der Trick besteht darin, dass man Skalarprodukte im Feature-Raum berechnet, indem man eine Funktion auf den Objekten im ursprünglichen Eingaberaum auswertet.

Dies hat mehrere Konsequenzen für die Praxis: man kann Kernfunktionen definieren, ohne die Abbildung ϕ in den Feature-Raum zu kennen, und damit Lernprobleme in einem Raum lösen, den man nicht explizit kennt. Allein durch Wahl eines anderen Kerns verschiebt man das Lernproblem in einen anderen Feature-Raum. Am wichtigsten für unsere Zwecke ist jedoch, dass man Kerne auch auf strukturierten Daten wie Graphen definieren kann [Sch97]: Man muss nur eine Funktion angeben, die die Ähnlichkeit zweier Graphen in Form eines Skalars misst, und zeigen, dass es sich dabei um einen gültigen Kern handelt (er muss positiv definit sein, siehe [SS02]).

Basierend auf Kernen wurden neben Support Vector Machines für Klassifikation und Regression zahlreiche weitere Kern-Verfahren (*kernel methods*) definiert, die nur in Form von Kernen auf die Daten zugreifen, zum Beispiel die Kernel-PCA [SS02].

3 Effiziente Graph-Kerne

Der naheliegende Ansatz, eine ganze Reihe von Algorithmen für die Analyse von Graphen zur Verfügung zu stellen, besteht also darin, eine ‘geeignete’ Kern-Funktion auf Graphen zu definieren. Um solch einen Kern zu definieren, müssen wir das in Abbildung 2 veranschaulichte Problem lösen: Gegeben zwei Graphen G und G', quantifiziere die Ähnlichkeit dieser Graphen in Form eines Skalars. Um für die Graphenanalyse geeignet zu sein, soll dieses Ähnlichkeitsmaß 3 Kriterien erfüllen: Erstens soll es die Struktur der Graphen vergleichen. Zweitens soll es dabei (falls vorhanden) die Attribute der Knoten und Kanten des Graphen berücksichtigen. Drittens soll es sich um einen gültigen Kern handeln, damit

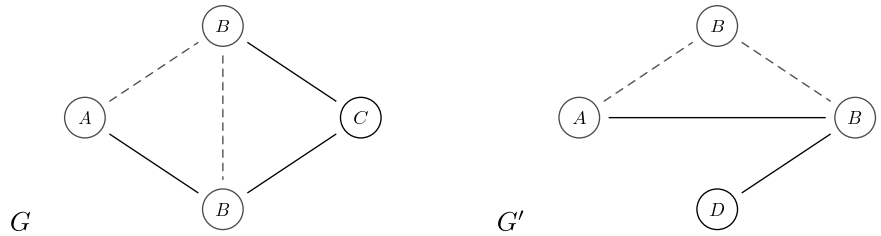

Abbildung 2: Graphenvergleich: Die Aufgabe beim Graphenvergleich besteht darin, die Ähnlichkeit der Graphen G und G' zu quantifizieren, wobei die Struktur und die Knotenattribute und Kantenattribute der Graphen berücksichtigt werden sollen. Der Random-Walk-Graph-Kern zählt gemeinsame Wege in zwei Graphen, wie z.B. die hier gestrichelt dargestellten Wege A-B-B.

es die Anwendung der zahlreichen Kern-Verfahren auf Graphen erlaubt.

Die ersten in der Literatur beschriebenen Graph-Kerne [GFW03, KTI03] basieren auf der Idee, gemeinsame Wege in zwei gegebenen Graphen G und G' zu zählen. Ein *Weg* ist dabei eine Sequenz von Knoten, die über Kanten miteinander verbunden sind. Im Gegensatz zu einem *Pfad* darf ein Weg denselben Knoten bzw. dieselbe Kante mehrfach enthalten.

Diese gemeinsamen Wege können mittels eines direkten Produktgraphen [GFW03] effizient bestimmt werden: Die Anzahl der gemeinsamen Wege von G und G' ist identisch zur Anzahl der Wege in ihrem direkten Produktgraphen $G_\times$ mit Knotenmenge $V_\times$ und Adjazenzmatrix $A_\times$. Der Produktgraph wird, anschaulich angesagt, aus der Menge aller Paare von Knoten und Kanten von G und G' gebildet. Wenn G und G' n Knoten enthalten, besitzt $G_\times$ daher $O(n^2)$ Knoten.

Da die Anzahl der Wege in einem nicht-leeren Graphen unendlich ist, müssen längere Wege mit einem Straffaktor $\lambda^k < 1$ versehen werden, wobei k die Länge des Wegs ist. Dann kann der Random-Walk-Kern (bei geeignter Wahl von λ) berechnet werden als:

$$k_\times(G, G') = \sum_{i,j=1}^{|V_\times|} [\sum_{k=0}^{\infty} \lambda^k A_\times^k]_{ij} = \sum_{i,j=1}^{|V_\times|} [(I - \lambda A_\times)^{-1}]_{ij} \tag{1}$$

wobei I die Einheitsmatrix ist.

Leider ist der Graph-Kern sehr langsam, da wir als einen Rechenschritt die $n^2 \times n^2$ Matrix $I - \lambda A_\times$ invertieren müssen — ein Aufwand von $O(n^6)$, der auf großen Graphen ($n > 30$) zu Laufzeitproblemen führt (siehe Abbildung 3).

Die wichtige Frage ist nun: Können wir den Random-Walk-Kern effizienter als in $O(n^6)$ berechnen, um ihn für größere Graphen brauchbar zu machen? Im ersten Teil meiner Doktorarbeit zeige ich [VBS07], dass man die Berechnung des Random-Walk-Graph-Kerns auf Sylvester-Gleichungen der Form

$$X = M + SXT \tag{2}$$

zurückführen kann, wobei S, M, T $n \times n$ Matrizen und gegeben sind und die $n \times n$ Matrix X bestimmt werden soll. Diese Gleichungen können in $O(n^3)$ Laufzeit gelöst werden.

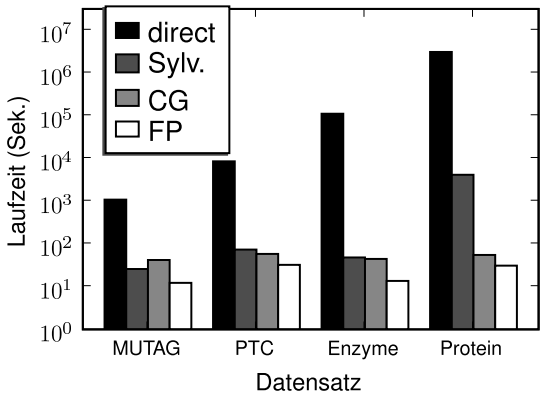

Abbildung 3: Laufzeit (in Sekunden auf einer log-Skala) für die Berechnung von 10^4 Kern-Werten auf vier Graphen-Datensätzen: MUTAG (durchschnittliche Knotenanzahl pro Graph: 18), PTC (27), Enzyme (33), Protein (39). Unsere neuen Ansätze basierend auf Sylvester-Gleichungen (Sylv.), konjugiertem Gradientenverfahren (CG) und Fixpunkt-Iteration (FP) sind bis zu Faktor 10^4 schneller als der bisherige Ansatz (direct).

In wenigen Schritten kann die Gleichung (2) in den Random-Walk-Kern in Gleichung (1) überführt werden. Damit wird es möglich, den Random-Walk-Kern exakt in $O(n^3)$ statt in $O(n^6)$ zu berechnen. Um die Tatsache auszunutzen, dass viele reale Graphen extrem *sparse* sind (d.h. wenige Kanten enthalten), kann man des Weiteren Gleichung (1) in ein Fixpunkt-Problem oder ein lineares Gleichungssystem umformen [VBS07] und dann mittels Fixpunkt-Iteration oder konjugiertem Gradientenverfahren lösen.

Auf Datensätzen aus der Bioinformatik und Chemoinformatik führen diese drei neuen Ansätze zu einer Beschleunigung um den Faktor 10^4 im Vergleich zur direkten Berechnung in $O(n^6)$, wie aus Abbildung 3 ersichtlich.

Auch unser beschleunigter Graph-Kern behebt jedoch noch nicht das sogenannte Wackel-Phänomen (*Tottering*) von Random-Walk-Kernen: Da Wege Wiederholungen von Kanten und Knoten erlauben, kann ein Random-Walk in zwei ungerichteten Graphen, die eine identische Kante $A - B$ enthalten, ständig zwischen A und B 'hin- und herwackeln'. Dadurch wird durch eine einzige gemeinsame Kante ein künstlich hoher Kern-Wert erzeugt.

Wie kann man Tottering vermeiden? Um Tottering auszuschalten, definieren wir einen Graph-Kern, der auf Pfaden statt Wegen beruht. Um nicht unter exponentieller Laufzeit zu leiden, betrachten wir die *kürzesten Pfade* [BK05]. Sie sind in $O(n^3)$ Laufzeit berechenbar, jedoch kann in Extremfällen die Anzahl der kürzesten Pfade in einem Graphen exponentiell mit seiner Knotenanzahl wachsen. Deswegen betrachten wir die Länge der kürzesten Pfaden in zwei Graphen - diese sind per Definition eindeutig und es gibt maximal n^2 solcher Distanzen in einem Graphen. In Experimenten ist dieser neue Graph-Kern i) schneller berechenbar als unsere schnellen Random-Walk-Kerne und ii) erreicht bessere Ergebnisse auf Graph-Klassifikationsproblemen.

Die letzte offene Frage ist: Während unsere neuen Graph-Kerne deutlich schneller sind als die bisher bekannten auf Graphen mittlerer Größe ($15 \leq n \leq 40$), kann man mit ihnen auch große Graphen mit $n >> 50$ Knoten vergleichen? Auf sehr großen Graphen sind die Random-Walk-Kerne (Laufzeit $O(n^3)$) und die Kürzeste-Pfad-Kern (Laufzeit $O(n^4)$) nicht effizient genug. Deswegen entwickeln wir einen weiteren Graph-Kern, den sogenannten *Graphlet-Kern*. Er basiert auf Ideen aus der Rekonstruktionstheorie für Graphen und Matrizen: Wir vergleichen zwei Graphen, indem wir bestimmen, wie häufig bestimmte kleine Subgraphen mit 4 Knoten, sogenannte *Graphlets*, in diesen Graphen auftreten.

Da die vollständige Aufzählung aller Graphlets $O(n^4)$ Laufzeit erfordert, wählen wir stattdessen die Strategie, Stichproben von Graphlets aus den Graphen zu ziehen. Erstmalig in der Literatur des *Subgraph Sampling* geben wir eine Formel für die erforderliche 'Stichprobengröße' an: Wie viele Graphlets müssen wir aus einem Graphen ziehen, um die Verteilung seiner Graphlets mit einer gewissen Präzision und mit einem gewissen Konfidenzlevel zu bestimmen? Interessanterweise ist diese Stichprobengröße *unabhängig* von der Anzahl der Knoten und Kanten des Graphen — dadurch wird dieser *Graphlet Kern* auch auf sehr große Graphen anwendbar, die bisher nicht mit Graph-Kernen handhabbar waren. Zudem erreicht er ähnlich gute Ergebnisse wie der Kürzeste-Pfad-Kern auf Klassifikationsproblemen.

4 Zwei-Stichproben-Tests auf Graphen

Bisher haben wir die Effizienz von Graph-Kernen verbessert, jedoch ein anderes Problem ausgeklammert: Graph-Kern-Werte an sich sind ein eher unintuitives Ähnlichkeitsmaß auf Graphen. Wenn wir zwei Graphen oder zwei Mengen von Graphen vergleichen, wird der (durchschnittliche) Kern-Wert groß sein, wenn die Graphen bzw. die Mengen von Graphen sehr ähnlich sind und andernfalls wird er klein sein. Nur wie kann man beurteilen, was ein *großer* Graph-Kern-Wert und was ein *kleiner* Graph-Kern-Wert ist?

Am besten wäre es, einen statistischen Test zu verwenden, um zu entscheiden, ob die Ähnlichkeit zweier Graphen signifikant ist. Die Frage, ob die Ähnlichkeit zweier Graphen signifikant ist, ist in der Literatur bisher kaum beachtet worden. Leider ist die Frage selbst ist schon problematisch: Was bedeutet die Aussage, dass die Ähnlichkeit zweier Graphen *statistisch signifikant* ist?

Für zwei Mengen von Graphen kann diese Frage leichter beantwortet werden als für zwei einzelne Graphen. Wenn wir zwei Mengen von Graphen X und Y untersuchen, können wir jede von diesen als eine Stichprobe einer zugrunde liegenden Wahrscheinlichkeitsverteilung (p bzw. q) betrachten. Dann müssen wir einen statistischen Test definieren, um zu entscheiden, ob die zugrunde liegenden Verteilungen dieser beiden Stichproben identisch sind ($p = q$). Dieses Problem wird in der Statistik als das Zwei-Stichproben-Problem bezeichnet. Ein zugehöriger Test ist ein Zwei-Stichproben-Test. Leider ist in der Statistik kein Zwei-Stichproben-Test für Mengen von Graphen bekannt.

Um solch einen Test zu definieren, müssen wir zunächst eine sinnvolle Test-Statistik definieren: unser Ansatz besteht darin, aus einer Menge von Funktionen $\mathcal{F}$ jene Funktion f

zu bestimmen, die auf der Stichprobe $X \sim q$ möglichst große Werte annimmt, und auf der anderen Stichprobe $Y \sim q$ möglichst kleine Werte (so negativ wie möglich). Unsere Test-Statistik soll die Differenz der Mittelwerte von f auf X und Y sein. Sollte dieser Wert groß sein, dann stammen die Stichproben wahrscheinlich aus verschiedenen Verteilungen. Wir bezeichnen diese Statistik als die Maximale Mittelwertsdiskrepanz (MMD, Englisch *Maximum Mean Discrepancy*) [BGR$^+$06, GBR$^+$07]. Analog können wir die Maximale Mittelwertsdiskrepanz zweier Verteilungen als die Differenz ihrer Erwartungswerte unter f definieren:

Definition 1 (Maximale Mittelwertsdiskrepanz) *Sei $\mathcal{F}$ eine Klasse von Funktionen $f : \mathcal{X} \to \mathbb{R}$. Dann ist die Maximale Mittelwertsdiskrepanz zweier Verteilungen p und q als*

$$\mathrm{MMD}(\mathcal{F}, p, q) := \sup_{f \in \mathcal{F}} \left(\mathbf{E}_{x \sim p}[f(x)] - \mathbf{E}_{y \sim q}[f(y)] \right)$$

und zweier Stichproben $X = \{x_i\}_{i=1}^{m_1}$ aus p und $Y = \{y_i\}_{i=1}^{m_2}$ aus q als

$$\mathrm{MMD}(\mathcal{F}, X, Y) := \sup_{f \in \mathcal{F}} \left(\frac{1}{m_1} \sum_{i=1}^{m_1} f(x_i) - \frac{1}{m_2} \sum_{i=1}^{m_2} f(y_i) \right).$$

definiert.

Wie gut MMD als eine Test-Statistik funktioniert, hängt davon ab, aus welcher Klasse $\mathcal{F}$ von Funktionen wir f bestimmen. Anschaulich gesagt, soll die Klasse groß genug sein, so dass wir auch kleine Unterschiede zwischen verschiedenen Verteilungen erkennen, gleichzeitig aber restriktiv genug, dass wir auch auf relativ kleinen Stichproben bereits identische Verteilungen erkennen.

Wir zeigen, dass eine bestimmte Klasse von Funktionen (genauer gesagt die Klasse der Funktionen aus der Einheitssphäre in einem universellen reproduzierenden Kern-Hilbertraum (RKHS) [Ste02]), diese Eigenschaften wunderbar erfüllt: $\mathrm{MMD}(\mathcal{F}, p, q) = 0 \Leftrightarrow p = q$, d.h. MMD ist nur Null, wenn die Verteilungen identisch sind. Daraus folgt eine weitere Beobachtung, die auch jenseits dieses Zwei-Stichproben-Tests bei der Entwicklung neuer Algorithmen und statistischer Tests von Bedeutung sein wird [GBR$^+$07]: *Jede Wahrscheinlichkeitsverteilung kann durch einen Punkt (ihren Erwartungswert) in einem universellen reproduzierenden Kern-Hilbert-Raum dargestellt werden.*

MMD in einem universellen Hilbert-Raum besitzt eine weitere für unsere Zwecke sehr günstige Eigenschaft. Wie wir in [BGR$^+$06] zeigen, kann MMD in Form von Kernen ausgedrückt werden:

Theorem 2 *Sei $\mathcal{F}$ die Einheitsspähre in einem RKHS $\mathcal{H}$ mit zugehörigem Kern k. Dann können $\mathrm{MMD}(\mathcal{F}, p, q)$ und $\mathrm{MMD}(\mathcal{F}, X, Y)$ folgendermassen berechnet werden:*

$$\mathrm{MMD}(\mathcal{F}, p, q) = \left[\mathbf{E}_{x,x' \sim p} \left[k(x, x') \right] - 2 \, \mathbf{E}_{x \sim p, x' \sim q} \left[k(x, x') \right] + \mathbf{E}_{x,x' \sim q} \left[k(x, x') \right] \right]^{\frac{1}{2}}$$

$$\mathrm{MMD}(\mathcal{F}, X, Y) = \left[\frac{1}{m_1^2} \sum_{i,j=1}^{m_1} k(x_i, x_j) - \frac{2}{m_1 m_2} \sum_{i,j=1}^{m_1, m_2} k(x_i, y_j) + \frac{1}{m_2^2} \sum_{i,j=1}^{m_2} k(y_i, y_j) \right]^{\frac{1}{2}}$$

Nachdem wir die Test-Statistik MMD mittels Kernen elegant berechnen können, müssen wir nun noch bestimmen, bis zu welchem Wert von MMD wir die Nullhypothese ($p = q$) annehmen. Ein einfacher Ansatz beruht auf Bootstrapping: Wir mischen beide Stichproben X und Y (und erzeugen damit eine gemeinsame Verteilung), splitten diese gemischte Stichprobe in zwei Hälften, berechnen MMD für diese Aufsplittung und wiederholen diesen Vorgang mehrfach. Wir erhalten eine Menge von MMD-Werten. Falls unser MMD-Wert unter den größten 5% dieser Werte liegt, können wir die Nullhypothese ($p = q$) auf Signifikanz-Niveau $\alpha = 0.05$ verwerfen.

In unseren Experimenten auf realen Daten schneidet MMD hervorragend ab: Auf vektoriellen Daten gelingt es MMD, mit geringerer Fehlerrate als bekannte Verfahren zu entscheiden, ob die zugrunde liegenden Verteilungen identisch sind oder nicht [GBR$^+$07]. Auch auf Graphen, auf die nur MMD anwendbar ist, erzielt MMD geringe Fehlerraten.

Mit MMD haben wir den ersten Zwei-Stichproben-Test auf Mengen von Graphen definiert. MMD kann auch auf zwei einzelne Graphen angewendet werden: In allen Graph-Kernen (siehe Abschnitt 3) werden Graphen als Mengen von Subgraphen dargestellt, seien es Mengen von Wegen, kürzesten Pfaden oder Graphlets. Wir können diese Mengen von Subgraphen als Stichproben auffassen und den Graphen, aus dem sie stammen, als die zugrunde liegende Verteilung. Auf diese Weise können wir MMD anwenden, um zu bestimmen, ob zwei Graphen statistisch signifikant ähnlich sind — nach dem MMD-Kriterium sind sie es, wenn ihre Subgraphen aus derselben Verteilung zu stammen scheinen.

Aus praktischer Sicht ist MMD zudem wegen seiner Effizienz attraktiv: Der Rechenaufwand ist nur quadratisch in der Größe der Stichproben ($O(m_1 + m_2)^2$).

5 Feature-Selektion auf Graphen

Die bisher vorgestellten Techniken beschäftigen sich mit der Frage: Wie ähnlich sind zwei Graphen? Genauso interessant ist die sich anschließende Frage: *Warum* sind zwei Graphen ähnlich? Welche *gemeinsamen* Subgraphen besitzen sie? In der Datenbank-Community kennt man letztere Fragestellung als das Problem der *Suche nach häufigen Subgraphen*.

Die Suche nach häufigen Subgraphen (*Frequent Subgraph Mining*) besteht darin, in einem Datensatz von Graphen $\{G_i\}$ Subgraphen S_j zu finden, so dass jeder dieser Subgraphen in mehr als t% der Graphen G_i enthalten ist. gSpan ist der bekannteste Ansatz, um diese häufigen Subgraphen zu bestimmen [YH02]: gSpan nutzt eine Branch-and-Bound-Suchstrategie, um den Suchraum einzuschränken, und eine effiziente Datenstruktur, den DFS-Code, um Subgraphen abzuspeichern und Mehrfachaufzählungen desselben Subgraphen zu vermeiden.

Wenn unser Datensatz $\{G_i\}$ allerdings ein Klassifikationsproblem darstellt, also wenn jeder Graph G_i einer Klasse Y_i zugeordnet ist, würden wir nicht nur gerne feststellen können, welche Subgraphen häufig im gesamten Datensatz sind, sondern welche Subgraphen *typisch* für bestimmte Klassen von Graphen sind und es uns erlauben, verschiedene Klassen von Graphen zu unterscheiden. Häufige Subgraphen sind dazu nur schlecht geeignet: Ein extrem häufiger Subgraph ist in fast allen Graphen anzutreffen und taugt nicht zur

Unterscheidung von Graph-Klassen. Sogar ein seltener Subgraph kann gleichmäßig über alle Klassen verteilt sein. Häufigkeit und Relevanz zur Klassenunterscheidung sind also nicht direkt korreliert.

Um wirklich *diskriminative* Subgraphen zu finden, benötigen wir Verfahren zur Feature-Selektion: Allgemein gesprochen, gegeben Objekte und ihre Klassenlabels wollen wir jene Features der Objekte bestimmen, die am stärksten mit den Klassenlabels korrelieren.

Zur Feature-Selektion auf häufigen Subgraphen entwickeln wir zunächst einen Algorithmus zur Feature-Selektion mittels Kernen auf allgemeinen (auch vektoriellen) Objekten und ihren Klassenlabels [SSG$^+$07]. Wir können diesen Ansatz jedoch nicht 1:1 auf Graphen anwenden, da die Anzahl der Subgraphen exponentiell mit der Größe der Graphen wächst.

Um unser Verfahren zur Feature-Selektion auf Graphen zu übertragen, integrieren wir es in gSpan [BYC$^+$08]. Auf diesem Weg erhalten wir ein Verfahren, das die Flexibilität von Kern-Verfahren mit der Effizienz von gSpan verbindet, um informative häufige Subgraphen zu bestimmen, die die Unterscheidung verschiedener Graph-Klassen ermöglichen. Nun können wir bereits während der Suche nach häufigen Subgraphen jene 'herausfiltern', die am stärksten mit der Klassenzugehörigkeit korrelieren. Im Vergleich mit den besten bekannten Verfahren für dieses Problem wählt unser Verfahren Subgraphen, die auf Klassifikationsproblemen bessere Vorhersagegenauigkeiten erreichen.

6 Zusammenfassung und Ausblick

In Form der hier vorgestellten Ähnlichkeitsmaße und Algorithmen auf Graphen steht Wissenschaftlern, die Netzwerke untersuchen, nun ein ganzer Katalog an neuen, effizienten und statistisch-fundierten Verfahren zur Datenanalyse auf Graphen zur Verfügung.

Der in Kapitel 4 vorgestellte Ansatz, Wahrscheinlichkeitsverteilungen durch ihren Mittelwert in einem speziellen Kern-Raum darzustellen, hat bereits zur Entwicklung weiterer neuer Algorithmen zur Datenintegration geführt [HSG$^+$07], und wir sind optimistisch, dass viele weitere folgen werden.

Literatur

[BGR$^+$06] K. M. Borgwardt, A. Gretton, M. J. Rasch, H.-P. Kriegel, B. Schölkopf und A. J. Smola. Integrating structured biological data by Kernel Maximum Mean Discrepancy. *Bioinformatics (ISMB)*, 22(14):e49–e57, 2006.

[BK05] K. M. Borgwardt und H.-P. Kriegel. Shortest-Path Kernels on Graphs. In *Proc. Intl. Conf. Data Mining*, Seiten 74–81, 2005.

[BYC$^+$08] K. Borgwardt, X. Yan, H. Cheng, L. Song, A. Gretton, A. Smola, H.-P. Kriegel, J. Han und P.S. Yu. Efficient Feature Selection in Frequent Subgraphs. Under preparation, 2008.

[GBR+07] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf und A. Smola. A Kernel Method for the Two-Sample-Problem. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007. MIT Press.

[GFW03] T. Gärtner, P.A. Flach und S. Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. In B. Schölkopf und M. K. Warmuth, Hrsg., *Proc. Annual Conf. Computational Learning Theory*, Seiten 129–143. Springer, 2003.

[HSG+07] J. Huang, A. Smola, A. Gretton, K. Borgwardt und B. Schölkopf. Correcting Sample Selection Bias by Unlabeled Data. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007. MIT Press.

[KTI03] H. Kashima, K. Tsuda und A. Inokuchi. Marginalized Kernels Between Labeled Graphs. In *Proc. Intl. Conf. Machine Learning*, Seiten 321–328, San Francisco, CA, 2003. Morgan Kaufmann.

[Sch97] B. Schölkopf. *Support Vector Learning*. R. Oldenbourg Verlag, Munich, 1997. Download: http://www.kernel-machines.org.

[SS02] B. Schölkopf und A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[SSG+07] L. Song, A. Smola, A. Gretton, K. Borgwardt und J. Bedo. Supervised Feature Selection via Dependence Estimation. In *International Conference on Machine Learning*, 2007.

[Ste02] I. Steinwart. Support Vector Machines are universally consistent. *J. Complexity*, 18:768–791, 2002.

[TC00] R. Todeschini und V. Consonni. *Handbook of molecular descriptors*. Wiley-VCH, 2000.

[Vap95] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[VBS07] S. V. N. Vishwanathan, Karsten Borgwardt und Nicol N. Schraudolph. Fast Computation of Graph Kernels. In B. Schölkopf, J. Platt und T. Hofmann, Hrsg., *Advances in Neural Information Processing Systems 19*, Cambridge MA, 2007. MIT Press.

[YH02] X. Yan und J. Han. gSpan: Graph-Based Substructure Pattern Mining. In *Proc. 2002 Int. Conf. on Data Mining (ICDM'02)*, Seiten 721–724, 2002.

Karsten Borgwardt wurde am 3. Dezember 1980 in Kaiserslautern geboren. Nach dem Abitur im Jahr 1999 am Justus-von-Liebig-Gymnasium in Neusäß in Bayern wurde er in die Bayerische Begabtenförderung (BayBFG) und die Stiftung Maximilianeum aufgenommen, während Studiums und Promotion ebenso in die Studienstiftung des deutschen Volkes. Er studierte Informatik an der Ludwig-Maximilians-Universität München (Diplom 2004) und Biologie an der Universität Oxford (M.Sc. 2003). Seine Diplomarbeit in Informatik verfasste er in der Statistical Machine Learning Group von NICTA Australia in Canberra, betreut von Dr. Alexander Smola. Von 1. Januar 2005 bis 22. Mai 2007 fertigte er an der LMU München seine Doktorarbeit mit dem Titel 'Graph Kernels' an, unter der Betreuung von Prof. Dr. Hans-Peter Kriegel. Seine Dissertation gewann im November 2007 den Heinz-Schwärtzel-Dissertationspreis. Seit September 2007 ist er Postdoctoral Research Associate an der Universität Cambridge, in der Machine Learning Group von Prof. Zoubin Ghahramani.