Über die Klassifizierung von Knoten in dynamischen Netzwerken mit Inhalt

Martin Thoma

Betreuer: Christopher Oßner

Zusammenfassung In dieser Arbeit wird der DYCOS-Algorithmus mit einer kleinen Modifikation vorgestellt. Er klassifiziert automatisch Knoten in Netzwerken, die bereits teilweise mit Labels versehen sind. Zur Klassifizierung kann er textuelle Informationen, die den Knoten zugeordnet sind, nutzen. Er ist auch für für viele 10 000 Knoten große, dynamische Netzwerke geeignet.

Keywords: DYCOS, Label Propagation, Knotenklassifizierung

1 Einleitung

1.1 Motivation

Teilweise gelabelte Netzwerke sind allgegenwärtig. Publikationsdatenbanken mit Publikationen als Knoten, Literaturverweisen und Zitaten als Kanten sowie Tags oder Kategorien als Labels; Wikipedia mit Artikeln als Knoten, Links als Kanten und Kategorien als Labels sowie soziale Netzwerke mit Eigenschaften der Benutzer als Labels sind drei Beispiele dafür. Häufig sind Labels nur teilweise vorhanden und es ist wünschenswert die fehlenden Labels zu ergänzen.

1.2 Problemstellung

Gegeben ist ein Graph, der teilweise gelabelt ist. Zustäzlich stehen zu einer Teilmenge der Knoten Texte bereit. Gesucht sind nun Labels für alle Knoten, die bisher noch nicht gelabelt sind.

Definition 1 (Knotenklassifierungsproblem) *Sei $G_t = (V_t, E_t, V_{L,t})$ ein gerichteter Graph, wobei V_t die Menge aller Knoten, E_t die Kantenmenge und $V_{L,t} \subseteq V_t$ die Menge der gelabelten Knoten jeweils zum Zeitpunkt t bezeichne. Außerdem sei L_t die Menge aller zum Zeitpunkt t vergebenen Labels und $f : V_{L,t} \to \mathcal{P}(L_t)$ die Funktion, die einen Knoten auf seine Labels abbildet.*

Weiter sei für jeden Knoten $v \in V$ eine (eventuell leere) Textmenge $T(v)$ gegeben.

Gesucht sind nun Labels für $V_t \setminus V_{L,t}$, also $\tilde{f} : V_t \to L_t$ mit $\tilde{f}|_{V_{L,t}} = f$.

1.3 Herausforderungen

Die Graphen, für die dieser Algorithmus konzipiert wurde, sind viele 10 000 Knoten groß und dynamisch. Das bedeutet, es kommen neue Knoten und eventuell auch neue Kanten hinzu bzw. Kanten oder Knoten werden entfernt. Außerdem stehen textuelle Inhalte zu den Knoten bereit, die bei der Klassifikation genutzt werden können. Das bedeutet, es ist wünschenswert bei kleinen Modifikationen nicht nochmals alles neu berechnen zu müssen, sondern basierend auf zuvor berechneten Labels die Klassifizierung modifizieren zu können.

2 DYCOS

2.1 Überblick

DYCOS (DYnamic Classification algorithm with cOntent and Structure) ist ein Knotenklassifizierungsalgorithmus, der Ursprünglich in [AgLi11] vorgestellt wurde. Er klassifiziert Knoten, indem mehrfach Random Walks startend bei dem zu

klassifizierenden Knoten gemacht werden und die Labels der besuchten Knoten gezählt werden. Das Label, das am häufigsten vorgekommen ist, wird zur Klassifizierung verwendet. Der DYCOS-Algorithmus nimmt jedoch nicht einfach den Graphen für dieses Verfahren, sondern erweitert ihn mit Hilfe der zur Verfügung stehenden Texte.

Für diese Erweiterung wird zuerst wird Vokabular W_t bestimmt, das charakteristisch für eine Knotengruppe ist. Wie das gemacht werden kann und warum nicht einfach jedes Wort in das Vokabular aufgenommen wird, wird in Abschnitt 2.5 erläutert.
Nach der Bestimmung des Vokabulars wird für jedes Wort im Vokabular ein Wortknoten zum Graphen hinzugefügt. Alle Knoten, die der Graph zuvor hatte, werden nun „Strukturknoten“ genannt. Ein Strukturknoten v wird genau dann mit einem Wortknoten $w \in W_t$ verbunden, wenn w in einem Text von v vorkommt.

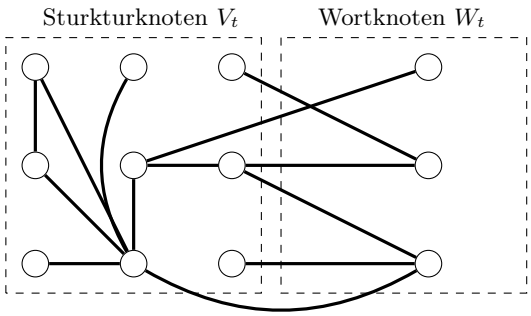

Abbildung 1. Erweiterter Graph

Der DYCOS-Algorithmus betrachtet die Texte, die einem Knoten zugeornet sind, als eine Multimenge von Wörtern. Das heißt, zum einen wird nicht auf die Reihenfolge der Wörter geachtet, zum anderen wird bei Texten eines Knotens nicht zwischen verschiedenen Texten unterschieden. Jedoch wird die Anzahl der Vorkommen jedes Wortes berücksichtigt.

2.2 Datenstrukturen

Zusätzlich zu dem gerichteten Graphen $G_t = (V_t, E_t, V_{L,t})$ verwaltet der DYCOS-Algorithmus zwei weitere Datenstrukturen:

– Für jeden Knoten $v \in V_t$ werden die vorkommenden Wörter, die auch im Vokabular W_t sind, und deren Anzahl gespeichert. Das könnte z. B. über ein assoziatives Array geschehen. Wörter, die nicht in Texten von v vorkommen, sind nicht im Array. Für alle vorkommenden Wörter ist der gespeicherte Wert

zum Schlüssel „Wort“ die Anzahl der Vorkommen von „Wort“ in den Texten von v.

– Für jedes Wort des Vokabulars W_t wird eine Liste von Knoten verwaltet, in deren Texten das Wort vorkommt.

2.3 Algorithmen

Bevor der Algorithmus formal beschrieben wird, wird eine Definition des strukturellen l-Sprungs benötigt:

Definition 2 *Sei $G_{E,t} = (V_t, E_{S,t} \cup E_{W,t}, V_{L,t}, W_t)$ der um die Wortknoten W_t erweiterte Graph.*

*Dann heißt ein Random Walk der Länge l in diesem Graphen ein **struktureller l-Sprung**, wenn für den Random Walk nur Kanten aus $E_{S,t}$ benutzt werden.*

Der strukturelle l-Sprung ist also ein Random Walk der Länge l im Graph G_t. Im Gegensatz dazu benötigt der inhaltliche l-Mehrfachsprung tatsächlich die Grapherweiterung:

Definition 3 *Sei $G_t = (V_t, E_{S,t} \cup E_{W,t}, V_{L,t}, W_t)$ der um die Wortknoten W_t erweiterte Graph.*

*Dann heißt ein Random Walk der Länge l in diesem Graphen ein **inhaltlicher l-Mehrfachsprung**, wenn für den Random Walk in jedem der l Schritte, startend von einem Knoten $v \in V_t$ eine Kante zu einem Wortknoten und von dem Wortknoten wieder zu einem Strukturknoten genommen wird.*

Algorithmus 1 DYCOS-Algorithmus

Input:
$\mathcal{G}_t = (\mathcal{N}_t, \mathcal{A}_t, \mathcal{T}_t)$ (Netzwerk),
r (Anzahl der Random Walks),
l (Länge eines Random Walks),
p_s (Wahrscheinlichkeit eines strukturellen Sprungs)
Output: Klassifikation von $\mathcal{N}_t \setminus \mathcal{T}_t$

```
function StrukturellerSprung(Dictionary d, Startknoten v, Länge l)
    for i von 1 bis l do
        v ← v.Next
        for each Label w in v.GetLabels do
            d[w] = d[w] + 1

function InhaltlicherMehrfachsprung(Dictionary d, Startknoten v, Länge l)
    for i von 1 bis l do
        v ← v.Next    ▷ TODO: Hier muss ein mehrfachsprung beschrieben werden!
        for each Label w in v.GetLabels do
            d[w] = d[w] + 1

for each Knoten v in N_t \ T_t do
    d ← Dictionary, das für neue Einträge 0 annimmt
    for i von 1 bis r do
        sprungTyp ← random(0, 1)
        if sprungTyp ≤ p_S then
            StrukturellerSprung(v, l)
        else
            InhaltlicherMehrfachsprung(v, l)
    if d ist leer then
        M_H ← HäufigsteLabelImGraph
        for each label in M_H do
            v.AddLabel(label)
    else
        M_H ← max(d)
        for each label in M_H do
            v.AddLabel(label)
return Labels für N_t \ T_t
```

2.4 Inhaltliche Mehrfachsprünge

Es ist nicht sinnvoll, direkt von einem strukturellem Knoten $v \in \mathcal{N}_t$ zu einem mit v verbundenen Wortknoten w zu springen und von diesem wieder zu einem verbundenem strutkurellem Knoten $v' \in \mathcal{N}_t$. Würde man dies machen, wäre zu befürchten, dass aufgrund von Homonymen die Qualität der Klassifizierung verringert wird. So hat „Brücke“ im Deutschen viele Bedeutungen. Gemeint sein

können z. B. das Bauwerk, das Entwurfsmuster der objektorientierten Programmierung oder ein Teil des Gehirns.

Deshalb wird für jeden Knoten v, von dem aus man einen inhaltlichen Mehrfachsprung machen will folgendes vorgehen gewählt:

1. Gehe alle in v startenden Random Walks der Länge 2 durch und erstelle eine Liste L, der erreichbaren Knoten v'. Speichere außerdem, durch wie viele Pfade diese Knoten v' jeweils erreichbar sind.
2. Betrachte im folgenden nur die Top-q Knoten, wobei $q \in \mathbb{N}$ eine zu wählende Konstante des Algorithmus ist.
3. Wähle mit Wahrscheinlichkeit $\frac{\textsc{Anzahl}(v')}{\sum_{w \in L} \textsc{Anzahl}(v')}$ den Knoten v' als Ziel des Mehrfachsprungs.

2.5 Vokabularbestimmung

Da die größe des Vokabulars die Datenmenge signifikant beeinflusst, liegt es in unserem Interesse so wenig Wörter wie möglich ins Vokabular aufzunehmen. Insbesondere sind Wörter nicht von Interesse, die in fast allen Texten vorkommen, wie im Deutschen z. B. „und“, „mit“ und die Pronomen.

Nun kann man manuell eine Liste von zu beachtenden Wörtern erstellen oder mit Hilfe des Gini-Koeffizienten automatisch ein Vokabular erstellen. Der Gini-Koeffizient ist ein statistisches Maß, das die Ungleichverteilung bewertet. Er ist immer im Intervall $[0, 1]$, wobei 0 einer Gleichverteilung entspricht und 1 der größt möglichen Ungleichverteilung.

Sei nun $n_i(w)$ die Häufigkeit des Wortes w in allen Texten mit dem i-ten Label.

darf ich hier im Nenner 1 addieren?

$$p_i(w) := \frac{n_i(w)}{\sum_{j=1}^{|\mathcal{L}_t|} n_j(w)} \qquad \text{(Relative Häufigkeit des Wortes } w\text{)} \qquad (1)$$

$$G(w) := \sum_{j=1}^{|\mathcal{L}_t|} p_j(w)^2 \qquad \text{(Gini-Koeffizient von } w\text{)} \qquad (2)$$

In diesem Fall ist $G(w) = 0$ nicht möglich, da zur Vokabularbestimmung nur Wörter betrachtet werden, die auch vorkommen.

Ein Vorschlag, wie die Vokabularbestimmung implementiert werden kann, ist als Pseudocode mit Algorithmus 2 gegeben. Dieser Algorithmus benötigt neben dem Speicher für den Graphen, die Texte sowie die m Vokabeln noch $\mathcal{O}(|\text{Verschiedene Wörter in } S_t| \cdot (|\mathcal{L}_t| + 1))$ Speicher. Die Average-Case Zeitkomplexität beträgt $\mathcal{O}(|\text{Wörter in } S_t|)$, wobei dazu die Vereinigung von Mengen M, N in $\mathcal{O}(\min |M|, |N|)$ sein muss.

Algorithmus 2 Vokabularbestimmung

Input:
$\mathcal{T}_t$ (Knoten mit Labels),
$\mathcal{L}_t$ (Labels),
$f : \mathcal{T}_t \to \mathcal{L}_t$ (Label-Funktion),
m (Gewünschte Vokabulargröße)
Output: $\mathcal{M}_t$ (Vokabular)

$S_t \leftarrow \textsc{Sample}(\mathcal{T}_t)$ ▷ Wähle eine Teilmenge $S_t \subseteq \mathcal{T}_t$ aus
$\mathcal{M}_t \leftarrow \bigcup_{v \in S_t} \textsc{getTextAsSet}(v)$ ▷ Menge aller Wörter
$cLabelWords \leftarrow (|\mathcal{L}_t| + 1) \times |\mathcal{M}_t|$-Array, mit 0en initialisert

for each $v \in \mathcal{T}_t$ **do** ▷ Gehe jeden Text Wort für Wort durch
  $i \leftarrow \textsc{getLabel}(v)$
  for each $(word, occurences) \in \textsc{getTextAsMultiset}(v)$ **do**
    $cLabelWords[i][word] \leftarrow cLabelWords[i][word] + occurences$
    $cLabelWords[i][|\mathcal{L}_t|] \leftarrow cLabelWords[i][|\mathcal{L}_t|] + occurences$

for each Wort $w \in \mathcal{M}_t$ **do**
  $p \leftarrow$ Array aus $|\mathcal{L}_t|$ Zahlen in $[0, 1]$
  for each Label $i \in \mathcal{L}_t$ **do**
    $p[i] \leftarrow \frac{cLabelWords[i][w]}{cLabelWords[i][|\mathcal{L}_t|]}$
  w.gini $\leftarrow \textsc{sum}(\textsc{map}(\textsc{square}, p))$
$\mathcal{M}_t \leftarrow \textsc{SortDescendingByGini}(\mathcal{M}_t)$
return $\textsc{Top}(\mathcal{M}_t, m)$

3 Experimentelle Analyse

3.1 DBLP

Im Folgenden wird die in [AgLi11] durchgeführte experimentelle Analyse des DYCOS-Algorithmus anhand des DBLP-Datensatzes (database systems and logic programming) der Universität Trier nachfolzogen. Dieser Datensatz beinhaltet bibliographische Daten von Informatik-Publikationen. Der Datensatz wird unter dblp.uni-trier.de/xml zur Verfügung gestellt. Die folgenden Informationen beziehen sich auf den Datensatz vom 28. Dezember 2013.

http://dumps.wikimedia.org/enwiki/20131202/ https://de.wikipedia.org/wiki/Hilfe:Download

4 Abgrenzung

Der in diesem Artikel vorgestellte DYCOS-Algorithmus wurde 2011 von Charu C. Aggarwal und Nan Li in [AgLi11] vorgestellt. Es gibt jedoch viele Klassifi-

zierungsalgorithmen, die ähnliche Ideen nutzen. Einige dieser Algorithmen werden im Folgenden kurz vorgestellt und Probleme, die der DYCOS-Algorithmus behebt, erläutert.

[BhCR07] nutzt nur die Struktur des Graphen zur Klassifizierung.

Der MUCCA-Algorithmus aus [Zapp] sei der erste. Dieser arbeitet auf gewichteten Graphen, im Gegensatz zu DYCOS.

5 Ausblick

Den sehr einfach aufgebauten DYCOS-Algorithmus kann man noch an vielen Punkten verbessern. So könnte man vor der Auswahl des Vokabulars jedes Wort auf den Wortstamm zurückführen. Dafür könnte zum Beispiel der Porter-Stemming-Algorithmus verwendet werden. Durch diese Maßnahme wird das Vokabular kleiner gehalten, mehr Artikel können mit einander durch Vokabular verbunden werden und der Gini-Koeffizient wird ein besseres Maß für die Gleichheit von Texten.

Wo steht was über den?

Eine weitere Verbesserungsmöglichkeit besteht in der textanalyse. Momentan ist diese noch sehr einfach gestrickt und ignoriert die Reihenfolge von Wortern beziehungsweise Wertungen davon. So könnte man den DYCOS-Algorithmus in einem sozialem Netzwerk verwenden wollen, in dem politische Parteiaffinität von einigen Mitgliedern angegeben wird um die Parteiaffinität der restlichen Mitglieder zu bestimmen. In diesem Fall macht es jedoch einen wichtigen Unterschied, ob jemand über eine Partei gutes oder schlechtes schreibt.

Eine einfache Erweiterung des DYCOS-Algorithmus wäre der Umgang mit mehreren Labels.

Literatur

AgLi11. Charu C. Aggarwal und Nan Li. On Node Classification in Dynamic Content-based Networks. In *SDM* [AgLi11], S. 355–366.

BhCR07. Smriti Bhagat, Graham Cormode und Irina Rozenbaum. Applying Link-Based Classification to Label Blogs. In Haizheng Zhang, Myra Spiliopoulou, Bamshad Mobasher, C. Lee Giles, Andrew McCallum, Olfa Nasraoui, Jaideep Srivastava und John Yen (Hrsg.), *WebKDD/SNA-KDD*, Band 5439 der *Lecture Notes in Computer Science*. Springer, 2007, S. 97–117.

Zapp. Giovanni Zappella. A Scalable Multiclass Algorithm for Node Classification.