Entwurfsbeschreibung - Deep Learning on Twitter

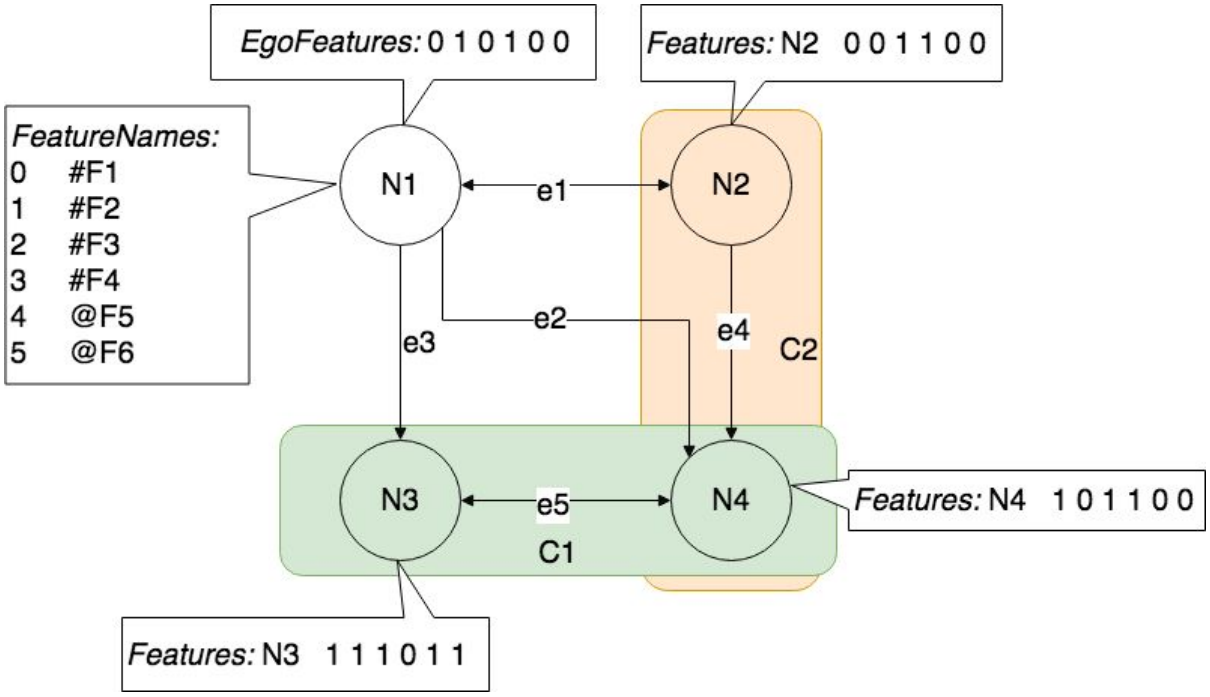
Das Ziel dieser Arbeit ist die Erstellung von Embeddings für einen beliebigen Graphen unter Verwendung von DeepLearning4j und den bereits implementierten DeepWalk[2] Algorithmus. Die generierten Embeddings sollen verwendet werden, um eine Webapplikation zu realisieren, die die Ähnlichkeiten zwischen Knoten darstellt. Die Darstellung soll dabei verschiedene Aspekte betrachten, wie z.B. Darstellung aller Knoten im Vektorraum, Ähnlichkeit zwischen Knoten, Top K Knoten bzgl. eines Knotens. Als Graph soll ein Social Network Graph verwendet werden. Weiterhin soll die Ähnlichkeit zwischen Nutzern mit den The- men verglichen werden für die sie sich interessieren. Diesbezüglich sollen die Fea- tures für jeden Nutzer verwendet werden und mithilfe des ParagraphVector[1] Modell ebenfalls als Embedding dargestellt werden. Die Eingabe für das Modell sind Dokumente. In dem Fall eine Aneinanderreihung der Tags, so dass für jeden Nutzer ein Embedding generiert werden kann

Inhaltsverzeichnis

Analyse der Eingabedaten

https://snap.stanford.edu/data/egonets-Twitter.html
http://i.stanford.edu/~julian/pdfs/nips2012.pdf

nodeId.edges : The edges in the ego network for the node 'nodeId'. Edges are undirected for facebook, and directed (a follows b) for twitter and gplus. The 'ego' node does not appear, but it is assumed that they follow every node id that appears in this file.
nodeId.circles : The set of circles for the ego node. Each line contains one circle, consisting of a series of node ids. The first entry in each line is the name of the circle.
nodeId.feat : The features for each of the nodes that appears in the edge file.
nodeId.egofeat : The features for the ego user.
nodeId.featnames : The names of each of the feature dimensions. Features are '1' if the user has this property in their profile, and '0' otherwise. This file has been anonymized for facebook users, since the names of the features would reveal private data.



Preprocessing

Ziel: Eingabedaten in ein einfach verarbeitbares Format konvertieren

Für das Preprocessing der Knoten und Kanten werden aus der Datei ‘twitter_combined.txt’ alle vorkommenden Knoten in eine Datei ‘combined_vertices.txt’ ausgegeben. Die Feature-Vektoren werden aus den ‘NodeID.egofeat’ und ‘NodeID.feat’ ausgelesen und für jeden Knoten jeweils einer Datei ausgegeben. Dabei werden die 0/1 Werte durch die Richtigen Feature Label ausgetauscht. Dadurch können die Featuredaten sehr einfach durch die Paragraph Vektoren gelesen und verwendet werden.

Graph initialisierung

- vertices aus datei combined_veritces.txt
- edges aus datei combined_edges.txt
- label mapping zur verwendung von internen inkrementellen ids
- zusätzliche kanten für degree=0 auf sich selber

DeepLearning4J

Das DeepLearning4J[1] framework dient zur Entwicklung dieser Aufgaben. Es realisiert eine Implementierung des DeepWalk und ParagraphVector Modells.

DeepWalk:

Ziel: Vorbereitete Knoten und Kanten laden, Graph erstellen, DeepWalk mit eigenen Windowsize und Walklength. Diese muss auf der internen Deeplearning4J-Repräsentation eines Graphen initialisiert werden, welcher aus den Dateien ‘combined_vertices.txt’ und ’combined_edges.txt’ erstellt wird. Der DeepWalk-Instanz werden bei Erstellung Werte für die Anzahl der Layer oder Größe des Windows übergeben. Diese Werte können bei Aufruf des Tools in der Kommandozeile konfiguriert werden. Dazu lässt sich noch die Länge der Walks konfigurieren auf denen die DeepWalk Embeddings generiert werden

ParagraphVectors:

Ziel: Vorbereitete Features Laden, Vektoren mit eigenen Windowsize erstellen

Die Feature Daten liegen für jeden Knoten als Datei vor, welche als Namen die ID des Knoten und als Inhalt eine durch Whitespaces getrennte Liste an Features besitzt.

Persistierung:

Ziel: Generierte DeepWalk und ParagraphVector Embeddings speichern und laden

Zur Persistierung werden die vorhanden Serialisierungen DeepLearning4Js genutzt um die geladenen DeepWalk und ParagraphVector Embeddings im Dateisystem abzuspeichern. Dazu gibt es auch passende Methoden, welche diese wieder laden.

Frontend

Für die Darstellung soll das Javascript Framework d3.js genutzt werden. Es soll eine globale Sicht der Knoten im dreidimensionalen Vektorraum, ein lokaler Vergleich zweier Knoten und eine top K Knoten Ansicht implementiert werden.
Für die globale Sicht werden alle Knoten geladen und aus Performanz-Gründen in einem zweidimensionalen Vektorraum dargestellt. Die Position wird dabei aus den Feature-Vektoren im jeweiligen Modell bestimmt. Die dritte Dimension wird über Farbe und

[1] https://deeplearning4j.org/

Größe des Knoten dargestellt. Knoten die einen niedrigeren Z-Wert aufweisen werden kleiner und roter dargestellt, während die Knoten mit einem hohen Z-Wert größer und blauer dargestellt werden.
Um schnelle Vergleiche zu ermöglichen gibt es zudem eine Option die Anzahl der geladenen Knoten zu begrenzen.

Für den lokalen Vergleich und die TopK-Ansicht werden zur Darstellung Force-Directed Graphen genutzt. Geladen werden die relevanten Knoten und die für das jeweilige Modell relevanten Features dieser Knoten. Übereinstimmende Features werden eingefärbt.
Bei der lokalen Vergleichssicht werden zwei Knoten-IDs eingegeben und zusätzlich zu den beiden Force-Graphen auch die berechneten Ähnlichkeitswerte angezeigt. Bei der TopK-Ansicht kann man eine Knoten-ID und k eingeben.

Die notwendigen Informationen werden über die REST-API abgefragt.

Anhang

Der Quellcode ist in folgendem Repository zu finden.

```
https://github.com/Termilion/Deep-Walk-4J
```

Referenzen

[1] Q. V. Le and T. Mikolov. Distributed Representations of Sentences and Documents. 32, 2014. https://cs.stanford.edu/~quocle/paragraph_vector.pdf

[2] B. Perozzi and S. Skiena. DeepWalk : Online Learning of Social Representations Categories and Subject Descriptors.
http://www.perozzi.net/publications/14_kdd_deepwalk.pdf