Beschreibung des Codes
Zu den drei Modellen

Florian Omiecienski

1 Beschreibung

In diesem Ordner sind die drei Modelle, welche in der Bachelor Arbeit beschrieben sind, umgesetzt. Der Ordner *./models/* enthält den python3-Code. Daneben liegen drei Programme, welche für die Durchführung der Experimente verwendet wurden.
Die Modelle und ihre Aufgaben sind in der Bachelor-Arbeit in Abschnitt *4. Methoden* beschrieben. Das Durchführen der Experimente erfordert die folgenden drei Schritte:

- n Modell-Instanzen trainieren (n-Fold-Cross-Validation)
- n Modelle auf n Splits testen und die Ergebnisse mitteln
- Auswerten der Ergebnisse

Zu diesem Zweck wurden drei python-Programme geschaffen. *experiment.py* führt eine 5-Fold-Cross-Validation durch und speichert die Daten-Splits, so wie die resultierenden Modell-Instanzen. *evaluate.py* evaluiert alle Modell-Instanzen auf den 5 Test-Splits und mittel die Ergebnisse. Diese werden in eine *pickle*-Datei gespeichert. *show_results.py* zeigt die Ergebnisse aus einer dieser Dateien in der Konsole und erstellt eine Reihe von Grafiken.

2 Code Struktur

Die drei Modelle sind in den Klassen Step1, Step2 und Step3 umgesetzt. Dies sind python-Objekte, die Methoden zu Verfügung stellen, um eine Modell-Instanz zu trainieren, um die Vorhersage-Schwellenwerte zu schätzen und um Daten vorherzusagen. Jede dieser Klassen verwendet eine korrespondierende Modell-Klasse (Model1, Model2, Model3). Das sind pytorch.nn.Module Objekte, welche über die Methode *forward* verfügen. Die drei Modell-Klassen sind aus einzelnen pytorch-Modulen zusammengesetzt. Alle pytorch-Module liegen in dem Ordner *./models/pytorch_models/*.
Im Ordner *./models/* liegen auch Dateien für die Klassen DataHandler, Evaluation und IndexExtractor. Die DataHandler-Klasse sammelt Methoden für die

Ein-/Ausgabe von Daten (z.B. laden der DebateNet-Daten, laden der Embedding-Vektoren, ...). Die IndexExtractor-Klasse wird verwendet um alle Worte, Buchstaben und Akteure auf Indices abzubilden. Diese Klasse wird benötigt, um ein trainiertes Modell zu verwenden. Die Evaluation-Klasse sammelt Methoden um die vorhergesagten Daten zu evaluieren.

3 Die Programme

Um die Experimente durchzuführen, wurde drei python3-Programme verwendet. Diese sind hier beschrieben.

experiment.py
Dieses Programm trainiert 5 Instanzen jedes Modells auf 5 unterschiedlichen Splits. Sowohl die Modell-Instanzen als auch die Splits werden dabei gespeichert. Anschließend kann man mit *evaluate.py* die Modelle testen.

evaluate.py
Dieses Programm lädt alle Modell Instanzen und Splits die zuvor erstellt wurden und testet die Modelle auf ihren jeweiligen Test-Splits. Die Ergebnisse werden über alles Splits gemittelt und in der Ausgabedatei *evaluation_results.bin* gespeichert.

show_evaluation.py
Dieses Programm lädt die Ausgabedatei von *evaluate.py* und zeigt ausgewählte Metriken in der Konsole an. Es ist möglich durch entfernen von Kommentaren in der Datei *show_evaluation.py* zusätzlich qualitative Beispiel und Precision-Recall-Kurven anzeigen zulassen.

4 Installations-Hinweise

Es werden folgende python3 Pakete benötigt:

- gzip
- json
- math
- matplotlib
- numpy
- os
- pickle
- random

- re
- torch[1]
- unicodedata

5 Benutzungs-Hinweis

Zum durchführen der Experimente, wie in der Bachelor-Arbeit beschrieben, muss nur das Skript *./experimente.sh* ausgeführt werden. Die Pfade sind in dieser Datei beschrieben und können dort auch geändert werden, wenn gewünscht. Bitte beachten Sie, dass mindestens 75 GB Speicherplatz benötigt werden, da 2*5*3 Modelle erstellt werden, die jeweils um die 2 GB an Speicherplatz brauchen. Zusätzlich werden 2*5 Kopien der DebateNet-Daten (Cross-Valiation-Splits) erstellt.
Die Ergebnis-Dateien in *./ergebnisse* und *./ergebnisse_random* sind in jeweils in eine zip-Datei verpackt und müssen erst entpackt werden um durch *show_results.py* angesehen werden zu können. Die DebateNet2.0-Daten liegen in einer zip-Datei vor und müssen ebenfalls entpackt werden um benützt zu werden. Das Passwort dieser zip-Datei ist auf der abgegebenen CD gespeichert.
Bitte beachten sie, dass mindestens 15-20 GB Arbeitsspeicher vorhanden sein müssen, da die FastText-Embeddings ca. 10 GB benötigen.
Vor Verwendung dieser Modelle müssen Akteur-Embeddings erstellt werden. Siehe dazu den Ordner */entity-embedding-bootstrap*.

[1] CUDA-Support empfohlen, aber nicht notwendig. Ohne kann die Laufzeit sehr lang werden.