A3, Anwendung, 3. Zwischenbericht

Philip Weinmann, phw8736@thi.de, Gruppe 1

20. Dezember 2021

Zusammenfassung

In diesem Zwischenbericht wird über das Trainieren eines Klassifikators zur Intenterkennung auf Basis des NLU-Datensatzes berichtet.

1 Auseinandersetzung mit der Problemstellung

Es ist für zukünftige Flugtaxen ein intelligenter persönlicher Assistent vorgesehen. Für diesen soll in diesem Projekt geprüft werden, ob und wie gut sich ein gegebener Datensatz als Trainingsdatensatz für NLU *(Natural Language Understanding)* eignet. Insbesondere soll in Richtung *Intent Detection* untersucht werden, das bedeutet, dass der intelligente Assistent anhand der natürlichen Sprache die Absichten des Gastes erkennt und so ein optimales Nutzererlebnis bieten kann.

2 Vorbereitende Schritte

2.1 Word Embedding

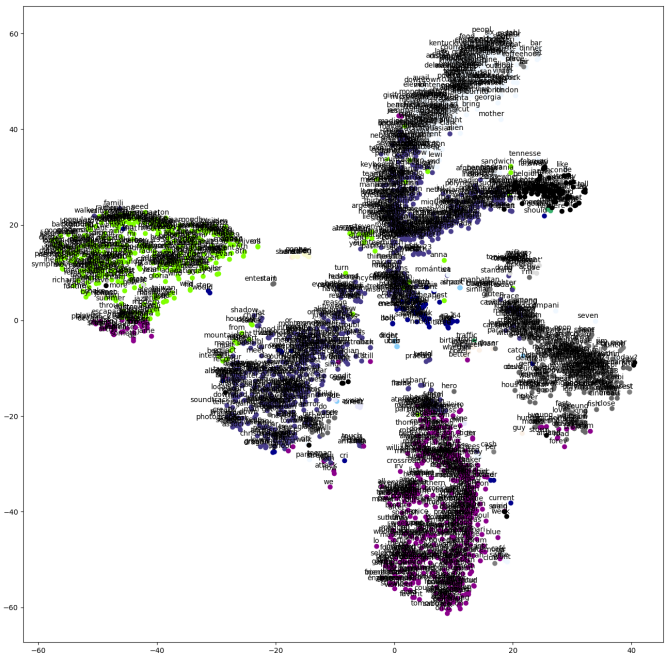
Für die geforderte *Intent Detection* wird *Word Embedding* eine nützliche Rolle spielen. *Word Embedding* bildet Wörter als Vektor ab. Im ersten Schritt wird das Wort hierbei durch das *One-Hot-Encoding*-Verfahren encoded. Der Vektor hat dabei die Länge des Vokabulars. Jeder Wortvektor besteht nach der Codierung ausschließlich aus Nullen, außer an genau einer, für jedes Wort einzigartigen, Stelle, wo eine 1 steht. Die Sammlung an Vektoren wird dann weiterverarbeitet, zunächst als Input für unser Embedding Modell. Durch die Multiplikation mit einer Matrix werden die Vektoren deutlich in ihrer Dimension reduziert (von der Länge des Vokabulars x 1 $\rightarrow$ ca. 200 x 1). Die *embedded words* dienen dann als Input für ein weiteres, anwendungsabhängiges Netz, werden durch eine neue Matrix decodiert und im einfachen Fall durch eine Softmax Funktion ausgegeben. In unserem Fall soll anhand der *embedded word vectors* die Intention bzw. die Absichten Autoren/Sprechers/... einzelner Sätze aus dem zugrunde liegenden Corpus vorhergesagt werden.

2.2 Textvorverarbeitung

Es ist für Word Embedding sehr wichtig für eine Normierung der Wörter im Corpus zu sorgen. Je unterschiedlicher Wörter mit gleicher semantischer Bedeutung geschrieben werden, desto schlechter wird unser Modell (z.B. BMW $\rightarrow$ B.M.W. $\rightarrow$ Bayrische Motorenwerke).

2.3 Umsetzung und Visualisierung

Das Modell wurde mit der Methode *fasttext.train_supervised()* trainiert und durch *sklearn.manifold.TSNE()* visualisiert. Hierbei wurde insbesondere an der Übergebenen Dimension(100-300), dem *minCount*, der Perplexität(10-50) und der *learning rate*(100-1000) ausprobiert. Untenstehend noch eine Visualisierung des optisch "bestenModells. Man erkennt die Clusterbildung von semantisch ähnlichen Wörtern.



3 Probleme bezüglich des NLU-Datensatzes

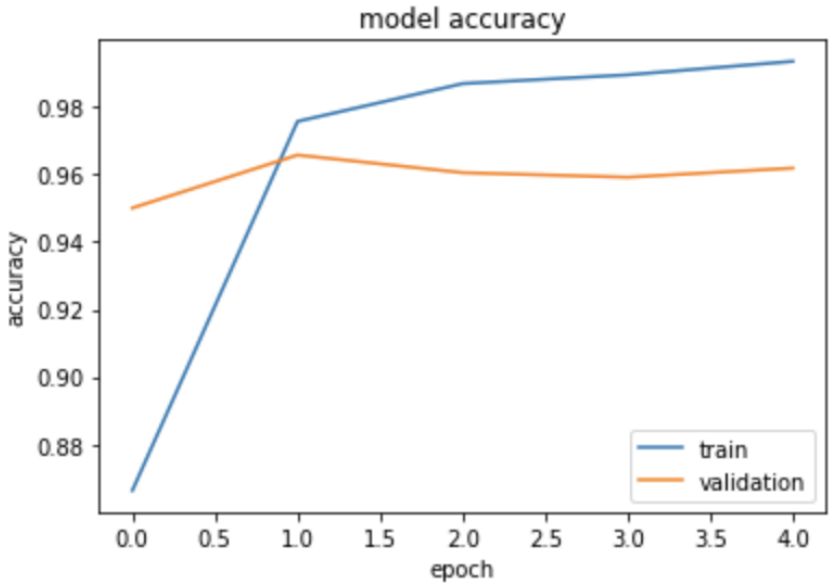
Während der Bearbeitung des Datensatzes ist aufgefallen, dass die Verteilung der Zielvariablen extram *imbalanced* ist. Das bedeutet, dass einige Ausprägungen teilweise über 100 mal häufiger in den Trainings- und Testdaten vorkommen als andere Ausprägungen. Ein Modell so zu trainieren ist nicht sinnvoll! Eine Methode um dagegen vorzugehen wäre beispielsweise, Samples aus der häufig vorkommenden Klasse zu entfernen. Da der Trainingsdatensatz bei einer Klassenanzahl von 15 allerdings nur aus 10000 Samples besteht, macht das in diesem Fall keinen Sinn. Der Datensatz würde mit dieser Methode zu klein werden. Die Counts der Samples pro Klasse ist in folgender Grafik gezeigt.

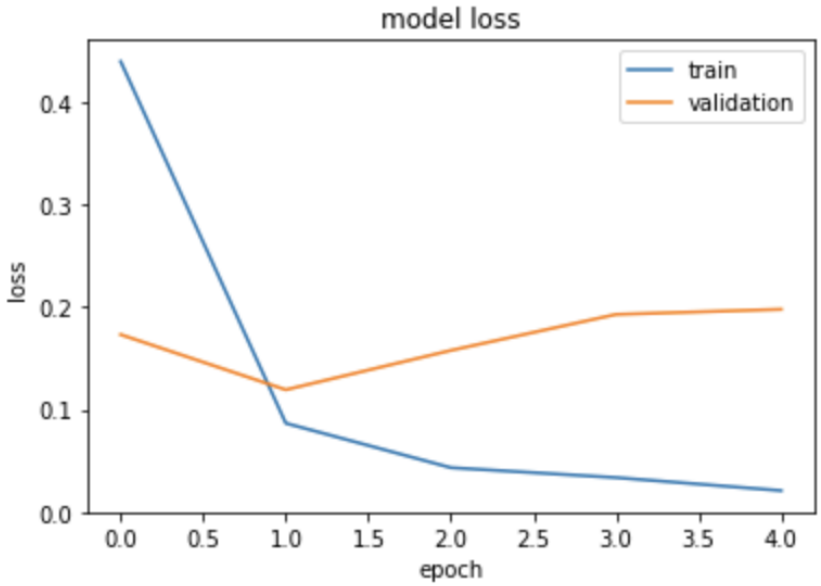
```
BookRestaurant 1508
PlayMusic 1460
GetWeather 1487
SearchScreeningEvent 1322
SearchCreativeWork 1347
AddToPlaylist 1359
RateBook 1389
GetTrafficInformation 10
GetDirections 18
SearchPlace 20
GetPlaceDetails 35
RequestRide 11
ComparePlaces 11
ShareETA 15
ShareCurrentLocation 8
```

4 Klassifikator

Zur Intent-Erkennung wurde ein neuer Klassifikator programmiert (siehe Python-Notebook). Dieser Klassifikator wurde durch ein *LSTM (long short-term memory)* umgesetzt. Ein LSTM-Netz erzielt insbesondere für sequentielle Eingaben unter Berücksichtigung von einigem Kontext sehr gute Ergebnisse. Das Modell wurde in der untenstehenden Grafik gezeigt aufgebaut. Die Embedded Vektoren wurden an die LSTM-Schicht übergeben. Nach drei *hidden* Schichten wurde per Softmax der *Intent* des eingegebenen Satzes vorhergesagt. Als Verlustfunktion wurde die Cross-Entropy gewählt, als Lernalgorithmus bzw. *Optimizer Nadam*. In den folgenden Grafiken eine Übersicht über die *Accuracy* und den *Loss* des Modells.

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_8 (Embedding)      (None, None, 200)         1895200

 lstm_8 (LSTM)                (None, 10)                8440

 dense_24 (Dense)             (None, 200)               2200

 dropout_17 (Dropout)         (None, 200)               0

 dense_25 (Dense)             (None, 200)               40200

 dense_26 (Dense)             (None, 15)                3015

=================================================================
Total params: 1,949,055
Trainable params: 1,949,055
Non-trainable params: 0
```





5 Variation der Modellparameter

Es wurden neben dem bereits genannten Klassifikator noch weitere Modelle getestet. Die Ergebnisse der Modelle mit den unterschiedlichen Einstellungen der Parameter ist in der untenstehenden Tabelle dargestellt. Alle der in der Tabelle stehenden Modelle wurden in 5 Epochen mit einer *Batch-size* von 32 trainiert.

Getestete Modelle							
Modell k	Anzahl Hidden Layer	Anzahl Neuronen Layer 1	Aktivierungs-funktion Layer 1	Anzahl Neu-ronen Layer 2	Aktivierungs-funktion Layer 2	Train-ACC	Test-ACC
0	2	200	Relu	200	Relu	99.53	96.10
1	2	200	tanh	200	tanh	99.79	96.39
2	2	200	Relu	100	Relu	99.67	95.50
3	2	200	Relu	500	Relu	99.43	95.36
4	1	200	tanh	-	-	99.82	96.15

Tabelle 1: Modellkonstellationen mit ACC-Scores für Trainings- und Validierungsdaten