Natural Language Processing 2021

Tutorial 3 — NLP21

Vector Semantics: Gensim, SpaCy, Word2Vec und GloVe

Nachdem Sie sich im letzten Tutorial mit semantischen Wortnetzen befasst haben, widmet sich dieses Tutorial *Vector Semantics*. Anstelle von manuell erzeugten synsemantischen Netzen, in denen Wörter in ist-Beziehungen miteinander verknüpft sind, bilden Vector Semantics automatisch über Kookkurrenzen in Korpora abgeleitete semantische Zusammenhänge ab.
Glücklicherweise gibt es Python Module, die das Arbeiten mit Word Embeddings stark vereinfachen und auch das Beziehen von vortrainierten Modellen unkompliziert ermöglichen.
In diesem Tutorial werden Sie mit *Gensim* arbeiten, einer Python Bibliothek speziell für das Arbeiten mit semantischen Vektoren (https://radimrehurek.com/gensim /index.html).
Bearbeiten Sie die Aufgaben in einem *Jupyter Notebook*.

Fabian Haak: fabian.haak@th-koeln.de

1. **Importieren der Module und Daten**.

   (a) **Import von NLP Modulen**.
   Importieren Sie wie im ersten Tutorial Pandas, Numpy, NLTK und RE.
   Installieren Sie via pip bzw. conda Gensim (in ihrer Konsole). Importieren Sie anschließend Gensim, KeyedVectors von gensim.models sowie den gensim.downloader wie folgt:

```
import gensim
from gensim.models import KeyedVectors
import gensim.downloader
```

   (b) **Modelle Downloaden und einrichten**.
   Sie haben in der Vorlesung von verschiedenen Attributen und Typen von Wort-Vektormodellen gehört, in diesem Tutorial werden Sie verschiedene Modelle miteinander vergleichen. Vortrainierte Modelle sind auf verschiedene Weise zu beziehen: Sie können über eigene Korpora selbst trainiert werden, über Vektoren bezogen werden (beispielsweise von github.com/ stanfordnlp/GloVe) und in Gensim in Modelle umgeformt werden oder, am unkompliziertesten, direkt über die von Bibliotheken wie Gensim nativ unterstützten vortrainierten Modelle abgerufen werden.
   Als Glove Modell haben wir Ihnen das "Glove 6b 100" Modell der genannten Quelle bereits vorbereitet. Sie müssen es lediglich über die "KeyedVectors" Klasse in Gensim als Modell laden:

```
glove_6b_100_model = KeyedVectors.load_word2vec_format(
"glove.6B.100d.w2vformat.txt6", binary=False)
```

   Lassen Sie sich als nächstes über den folgenden Befehl eine Liste aller in Gensim enthaltenen Modelle ausgeben:

```
print(list(gensim.downloader.info()["models"].keys()))
```

   Laden Sie das "glove-twitter-100" Modell über "gensim.downloader.load()" wie folgt:

```
glove_twitter100_vectors = gensim.downloader.load("glove-twitter-100")
```

   $\longrightarrow$ **Achtung, der Vorgang kann Zeit in Anspruch nehmen und beansprucht viel Arbeitsspeicher, sollten Sie Probleme haben, überspringen Sie das Downloaden des vortrainierten Modells über Gensim!**

Fabian Haak: fabian.haak@th-koeln.de

Die letzte hier vorgestellte Möglichkeit, zu Vektormodellen zu gelangen, ist das Erzeugen eines Modells über einen Korpus aus Texten. Beispielhaft gehen Sie bitte wie folgt vor:

```
corpus = gensim.downloader.load('text8')
from gensim.models.word2vec import Word2Vec
word2vec = Word2Vec(corpus)
```

2. **Funktionen von Vector Semantics.**
   Sie sollten nun zwischen einem und drei verschiedenen Vektormodellen geladen haben. Viele vektorsemantische Operationen lassen sich über die Word2Vec Modul API von Gensim (HIER abzurufen)) anwenden. Lassen Sie sich von der Namensgebung nicht verwirren, die Methoden sind allgemein für Vektormodelle anwendbar, nicht nur für Word2Vec.

   (a) **Ähnliche Ausdrücke finden.**
   Bestimmen Sie per *similar_by_word()* methode die 10 ähnlichsten Begriffe zu den Wörtern "summer", "salad" und "python". Diskutieren Sie in der Gruppe: Was fällt Ihnen auf? Können Sie Unterschiede zwischen den Modellen, die auf größeren Korpora basieren (glove.6b und glove_twitter100) und dem von Ihnen auf Basis des 32MB Korpus trainierten Word2Vec Modell feststellen? Wie könnten die Unterschiede zustande kommen?

   (b) **Relational Similarity.**
   Mit den Vektorsemantischen Modellen lassen sich Analogien der Form "A verhält sich zu B, wie A* zu ...?", wie diese in den aus den aus dem Korpus extrahierten Informationen vorliegen, ableiten. Mithilfe der Methode most_similar() (siehe in Einleitung zur Frage 2 verlinkte Gensim Word2Vec Modul API) können Sie so z.B. über "Paris -France +Italy" "Rom" finden lassen:

```
print('glove: Paris -France +Italy: ',
glove_6b_100_model.most_similar(positive=["paris", "italy"],
negative=["france"], topn=3))
```

   So kann auch in dem Korpus enthaltener Bias aufgedeckt werden. Für dichotome Merkmale wie [man,woman] oder [young,old] setzen Sie jeweils eines der Merkmale in "positive" und "negative" ein. Finden Sie heraus, was die Modelle für "doctor +woman", "housewife +man" sowie "car +sea" als 3 nächste Assoziationen enthalten, wählen Sie dabei sinnvolle negatives. Diskutieren Sie in der Gruppe! Finden Sie noch weitere relationale Symmetrien?

   (c) **Bonus: (Sentence) Similarity.**
   Die Methode "n_similarity([], [])" kann genutzt werden, um die Ähnlichkeit zweier Listen von Tokens zueinander zu beurteilen. Fällt Ihnen ein sinnvolles Anwendungsbeispiel ein? Diskutieren und probieren Sie!

3. **Deutsche vortrainierte Vektormodelle mit SpaCy.**

   (a) **Module und Daten importieren.**
   Installieren Sie zunächst SpaCy (über pip/conda in ihrer Konsole). Laden Sie anschließend ein Modell für deutsche Sprache ebenfalls in ihrer Konsole/Terminal über die folgende Eingabe herunter:

   ```
   python -m spacy download de_core_news_lg
   ```

   Achtung: Vermutlich müssen Sie ihr Jupyter Kernel neu starten, damit ihr System das Modell finden kann. Laden Sie in ihrem Notebook das Modell mit

   ```
   nlp = spacy.load('de_core_news_lg')
   ```

   (b) **Vector Semantics in SpaCy.**
   SpaCy ist ein mächtiges, leicht zu bedienendes NLP Werkzeug. Mit einem Aufruf von nlp(*TEXT*) (wenn Sie Ihr Modell beim Laden so genannt haben) wird der Text tokenisiert, lemmatisiert und Wortvektoren und viele weitere Features geladen wenn das Modell entsprechende Informationen enthält. Dazu kommen Komfortfeatures wie verschiedene Visualisierungsoptionen. Lassen Sie sich mit dem folgenden Code für verschiedene Sätze Ihrer Wahl die von Spacy erkannten PoS und Struktur Tags ausgeben:

   ```
   satz = nlp(HIER IHR SATZ ALS STRING)
   displacy.render(satz, style='dep')
   ```

   (c) **Similarity mit SpaCy.**
   Laden Sie über Pandas read_csv() die Liste von single_term_suggestions.txt aus Tutorial 2 und speichern Sie diese als Dataframe. Fügen Sie eine weitere Spalte "city" hinzu, in der Sie die die *Similarity* des Begriffs (suggestion_ger) zu nlp("Stadt") bestimmen. Gehen Sie analog hierzu für "Politik" und "Freizeit" vor und Speichern Sie die Similarities in sinnvoll benannten Spalten. Diskutieren Sie in der Gruppe: Wofür könnten Sie die bestimmten Similarities nutzen?