Tutorial Information Retrieval

Universität Innsbruck - Department of Computer Science

E. Zangerle

2019-05-08

Exercise Sheet 5

Exercise 1 (Word Embeddings Theory) [3 Points]

This week, we are going to explore word embeddings, a state-of-the-art vector representation method for words that also captures the context in which these words occur. This allows to not only perform retrieval tasks based on syntactic but also on semantic similarity.
To get a first overview, please watch Chris Manning's introductory lecture on distributed word vector representations and word2vec (first 42 minutes of the video)[1]. Answer the following questions regarding distributed vector representations:

a) *1.5 Points* What is the basic intuition behind word2vec and similar methods?

b) *1.5 Points* How are those distributed representations of words computed?

For further information, there's also a short talk by Tomáš Mikolov about word2vec[2] and naturally, numerous blogs and articles on this topic. You might also be interested in Mikolov's original paper:

Exercise 2 (Word/Document Embeddings Hands-On) [7 Points]

The `gensim` python module provides a highly efficient implementation of the word2vec model (and a number of further language models including topic models such as LSI or LDA, but also doc2vec...). We are going to work with gensim in the following.
In principle, there are two ways to gather embeddings for a specific retrieval task: either we rely on pre-computed embeddings (e.g., utilizing the widely used GoogleNews vectors[3]) or, to provide a more customized and corpus-specific embedding, we can compute the embeddings ourselves for a given text corpus.

a) *1 Point* To explore embeddings that were computed by word2vec, we firstly rely on pre-trained models. Retrieve and load such a model from e.g., Google (see above, 1.3G) or via SpaCy[4] (four different models of different sizes for English available). Subsequently, look into how such a model can be utilized to find similar and related words. Come up with a number of example queries that show that word2vec indeed captures the semantic relatedness of words.

b) *2.5 Points* In the second step, we further explore semantic relatedness between words by plotting related words in a 2d representation. The pre-trained models are of high dimensionality

[1] Lecture 2 | Word Vector Representations: word2vec, Stanford School of Engineering, Chris Manning
[2] Distributed Representations for NLP (Machine Learning Prague 2016), Tomáš Mikolov
[3] word2vec: Pre-trained word and phrase vectors, Google
[4] SpaCy: Available pre-trained statistical models for English

(e.g., 300 dimensions for the GoogleNews vectors), hence, to provide a 2-dimensional representation, we have to apply a dimension reduction method to map the embedding vectors to the 2-dimensional space. We require this mapping method to preserve neighbors—i.e., we require that neighbors in the high-dimensional space are also neighbors in the 2d space. For this task, you can make use of e.g., Principal Component Analysis, Uniform Manifold Approximation, and Projection (UMAP) or t-distributed stochastic neighbor embedding (t-SNE), all of which are available for Python (with t-SNE and PCA available in scikit-learn). Plot a sample of the words contained in the 2d space and inspect their actual similarity (e.g., by searching for synonyms or related concepts such as the legendary king and queen example [5]).

c) *1 Point* Related to word2vec is the so-called doc2vec representation, which allows to not only describe words but a full document by a high-dimensional vector representation. Utilize gensim to compute the doc2vec representation of the Reuters_50_50 dataset we already used on the last sheet. How does doc2vec differ from word2vec in terms of the representation computed and the actual computation?

d) *2.5 Points* Based on the document vectors computed in the previous step, we aim to cluster documents that are related in the following to . To do so, we will utilize the widely used k-means clustering algorithm. To apply k-means, you have to specify the number of clusters to be computed. How does this choice influence the result of the resulting clustering? Please experiment with different numbers of clusters and look into how k-means computes those clusters. Please also visualize the clusters you computed in 2d space and visually inspect the quality of the clustering.

Important: Submit your solution to OLAT and mark your solved exercises with the provided checkboxes. The deadline ends at 23:59 on the day before the discussion.

[5] King - Man + Woman = Queen: The Marvelous Mathematics of Computational Linguistics