## Feedback for WS4

We only received 4 submissions for the fourth worksheet. Most probably you preferred to concentrate on your assignment but please ensure that you go over this week's course material. Word embeddings are important and widely used in downstream NLP tasks.

#### [E1]:

The formulas for the computation of the mentioned concepts (cosine similarity, MLE, PPMI based cosine similarity etc.) are given in your text book. You can write a Python script to find the outcome of the formulas for the given values. It would be interesting to see how different are the computed values in item (1) and (4). It is worth to think about potential reasons if the computed values are significantly different from each other.

#### [E2]:

The concept "woman" can be understood as a 2 dimensional vector in the given scheme, where, for example, the x-axis describe the "human" property while the y-axis describe the "gender" property. "Man" can be represented as human:yes, gender: male. When describing the vector in Euclidean-space, we can use x = 1 if human and x = 0 otherwise, y = 1 if gender is equal to female and y = 0 if gender is male. More concepts can be represented by adding features, and therefore by increasing the dimension of the vector.

The cosine similarity tracks semantic closeness to a certain extent, as the vector with more features will be sparse, it may fail in capturing the similarity between words. The advantage of such system is that it is easily interpretable by humans. However, adding more concepts does not scale too well with the amounts of weight that must be learned by the model in order to capture the relationship. You can also imagine how manual construction of such a knowledge base can fail in covering all the necessary concepts and be error prone considering the number of features included in the model.

#### [E3]:

A (static) word embedding is a function that maps each word type to a single vector. No matter how many senses a word has, word embedding method embeds all senses into a single vector representation. So the (static) word embeddings cannot encode context, and this can cause a problem for downstream NLP tasks like document classification.

According to [word embedding demo] (http://bionlp-www.utu.fi/wv\_demo/), using a model trained on Google News, the top 5 nearest words of bank in vector space are banks, banking, Bank, lender and banker. We can see that the semantic of bank here means "financial institution" and does not capture the meaning of a slope land besides water. Therefore, the text “The man went fishing by the bank of the river.” would be misclassified to “Financial institution” class in document classification because the word vector of bank points to the meaning of a financial institution.

In recent years, context is integrated into word embeddings such as provided by ELMo (Peters et al.,2018), or BERT (Devlin et al., 2019). You can refer to the relevant papers to check how these contextualized word embedding’s provide semantic vector representations of words depending on their respective context.

##### References:

1. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 4171–4186, Minneapolis, MN, USA.
2. Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 2227–2237, New Orleans, LA, USA.

#### [E4]:

For the exploration of functions provided by the gensim library, you can do vector operations (see below), calculate similarity between words etc.

* print(model.most\_similar(positive=['woman', 'king'], negative=['man']))
* print(model.most\_similar(positive=['woman', 'doctor'], negative=['man']))
* print(model.doesnt\_match("breakfast cereal dinner lunch".split()))
* print(model.similarity('woman', 'man'))

These kinds of experiments can be interesting to see whether there is a gender, ethnical etc. bias in the system's responses. The provided additional file (wordsim\_similarity\_goldstandard.txt) includes human judgements about word similarities. It would be also interesting to compute the similarities in that file by the model and compare the results with the human judgements.

#### [E5]:

One simple implementation of paraphraser can be done by replacing the words in the input string with the ones similar to them. For that purpose, you can compute the cosine similarities with the **most\_similar** function provided by the gensim library.

You can play with the threshold of the acceptable cosine similarity to improve the quality of the generated outcome. When we set the similarity threshold as 0.8, our sample implementation re-wrote the sentence "This model is not very intelligent" as "It model is not extremely intelligent" which clearly suggest that there is a room for improvement in our sample implementation.

#### [E6]:

The method evaluate\_word\_analogies can be used to evaluate the outcome of the model by comparing it the provided analogies. The incorrect constructions can be extracted from the outcome and closer investigation of the mis-related words can be helpful to recognize some patterns if any.

#### [E7]:

You can consider following questions to figure out the potential disadvantages of using mentioned word embeddings:

1. Can they handle unknown or out-of-vocabulary (OOV) words?
2. Can they encode morphological relations (i.e., are there sub-word level representations)?
3. Can they handle polysemy, homonymy and synonymy when needed?