**Bachelor thesis – methods**

Alle Links sind zwischen dem 06.08 und dem 08.08 abgerufen worden.

Docker:

* <https://de.wikipedia.org/wiki/Docker_(Software)>
* Isolate applications in container
* Container: instance of image. Is stopped when program finishes.
* Image: consists of non-writeable layers. Multiple containers can be started from one image.
* Container has one writable layer, in which it differs from other instances of image.
* Needs Linux kernel.

Soft Cosine Measure (SCM):

* <https://radimrehurek.com/gensim/auto_examples/tutorials/run_scm.html#sphx-glr-auto-examples-tutorials-run-scm-py>
* Compute semantic similarity between documents using inner\_product method
* Uses Word2Vec and bag-of-words vector representation of documents (i.e. word’s frequencies)
* Assumption: document vectors -> non-orthogonal basis
* Angle between two basis vectors = angle between word2vec embeddings of words
* Uses genism.similarities.[SparseTermSimilarityMatrix][WordEmbeddingSimilarityIndex]
  + Index initiated with trained/ downloaded model
  + Matrix initiated with index, dictionary and tfidf
  + Similarity via inner product of matrix of sentences and normalized

Word Mover’s Distance (WMD):

* <https://radimrehurek.com/gensim/auto_examples/tutorials/run_wmd.html#sphx-glr-auto-examples-tutorials-run-wmd-py>
* Query for documents
* Uses Word2Vec
* Outperforms knn classification
* Find minimum travelling distance (i.e. most efficient way to move distribution of document 1 to distribution of document 2)

Latent Dirichlet Allocation (LDA):

* <https://towardsdatascience.com/end-to-end-topic-modeling-in-python-latent-dirichlet-allocation-lda-35ce4ed6b3e0> (GOOD guide incl. Code)
  + Topic: cluster of words
  + Text: mixture of all topics with specific weight
  + Alternatives for topic modelling: Latent Semantic Analysis (LSA/LSI), Probabilistic Latent Semantic Analysis (pLSA)
* How to train LDA model: <https://radimrehurek.com/gensim/auto_examples/tutorials/run_lda.html#sphx-glr-auto-examples-tutorials-run-lda-py>
* Components of LDA model: <https://thinkinfi.com/latent-dirichlet-allocation-for-beginners-a-high-level-overview/>
* <https://thinkinfi.com/latent-dirichlet-allocation-explained/> (GOOD explanation)
  + Matrix: document id = y-axis, token position = x-axis, entry = token/ id of token
  + Randomly assign topic to tokens in matrix (same token can have different topics)
  + Calculate Count matrix
    - Topic count matrix:
      * Count number of times a token is assigned to a topic
    - Calculate Document-topic matrix
      * Entry = number of tokens assigned to the topic for the document
  + Collapse Gibbs Sampling
    - Calculates probability of topic for a word in a document
    - Complex formulae
    - Multiple iterations, recalculating probability of a word being from a topic -> topic is highest probability

Embeddings from Language Models (ELMo):

* <https://www.geeksforgeeks.org/overview-of-word-embedding-using-embeddings-from-language-models-elmo/>
* Creation via methods, such as Word2Vec, CBOW, Skip Gram, Glove, Elmo, One Hot encoding, TF-IDF, FastText etc.
* ELMo word vectors:
  + Two-layer bidirectional language model (biLM)
  + Each layer: forward and backward pass
  + Word representation using the complete sentence -> captures context

Pre-trained Word embeddings using Glove:

* <https://www.geeksforgeeks.org/pre-trained-word-embedding-using-glove-in-nlp-models/>
* Global vectors = Glove
* Pre-defined dense vector dictionaries for four different dimensionalities

Word similarity using spaCy:

* <https://www.geeksforgeeks.org/python-word-similarity-using-spacy/>
* Similarity in [0,1] = semantically closeness = similarity of vectors in vector space
* Similarity via context-sensitive tensors or word vectors

Pretrained model by Google:

* https://www.geeksforgeeks.org/finding-the-word-analogy-from-given-words-using-word2vec-embeddings/
* Usable with Word2Vec as predefined vocabulary
* Download pre-trained embeddings/model via: genism.downloader.api.load(‘word2vec-google-news-300’) (<https://radimrehurek.com/gensim/auto_examples/tutorials/run_scm.html#sphx-glr-auto-examples-tutorials-run-scm-py>)

FastText Model:

* <https://radimrehurek.com/gensim/auto_examples/tutorials/run_fasttext.html#sphx-glr-auto-examples-tutorials-run-fasttext-py>
* Belief: morphological structure of word is important
  + != Word2Vec (trains unique embedding for every individual word)
  + E.g. for German: single word can have a lot of morphological forms
* Word = aggregation of subwords = character ngrams of a word
* Vector of word = sum of all vectors of its component char-ngrams
* Small Training size + syntactic task: better than Word2Vec
* Semantic task: worse than Word2Vec
* can obtain vectors even for out-of-vocabulary (OOV) words
  + if >= 1 char-ngrams was in training data -> sum those up
* <https://medium.com/analytics-vidhya/sentence-embeddings-facebooks-infersent-6ac4a9fc2001>
  + Word embedding != sentence embedding
  + Baseline of ELMo

Annoy:

* <https://radimrehurek.com/gensim/auto_examples/tutorials/run_annoy.html#sphx-glr-auto-examples-tutorials-run-annoy-py>
* library for similarity queries on top of vectors learned by Word2Vec
* Approximate Nearest Neighbours Oh Yeah = Annoy
* Finds approximate nearest neighbours in sub-linear time (fast than knn)
* Uses trees (more trees = more accurate)
* indices can be memory-mapped from disk

Word2Vec:

* <https://www.alexanderthamm.com/de/data-science-glossar/word2vec/>
  + Neuronal network
  + Text analysis via embedding
  + Numerical vectors used for context
* <https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/>
  + Word embeddings generated by neural networks, co-occurrence matrix, probabilistic models, etc.
  + Word2Vec: shallow two-layer neural networks (input, hidden and output layer)
    - CBOW (Continuous Bag of Words):
      * Bag -> order of words in context not relevant (https://www.alexanderthamm.com/de/data-science-glossar/word2vec/)
      * uses context to predict word
      * input layer: context words
      * output layer: current word
      * hidden layer: #dimensions = dimension to present current word at output layer
      * good at syntactic relationship (https://www.alexanderthamm.com/de/data-science-glossar/word2vec/)
    - Skip Gram
      * Predicts context words
      * input layer: current word
      * output layer: context words
      * hidden layer: #dimensions = dimension to present current word at input layer
      * good at semantic relationship (https://www.alexanderthamm.com/de/data-science-glossar/word2vec/)
    - words in similar context ~ close in vector space
* Efficient Estimation of Word Representations in Vector Space (Paper)

Bag-of-words

* <https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html#sphx-glr-auto-examples-tutorials-run-doc2vec-lee-py>
* Transforms document/ sentence into fixed-length vector of integers
* Element of vector: counts number of times a word occurred
* Oder of elements is arbitrary -> information about order of words is lost
  + Bag of n-grams (word phrases of length n): capture word order, but high data sparsity and high dimensionality
* Does not learn meaning of words
  + != Word2Vec
* Function: genism.dictionary.doc2bow (https://tedboy.github.io/nlps/generated/generated/gensim.corpora.Dictionary.doc2bow.html)
  + Returns list of (token\_id, token\_count)
  + Assumption: tokens are tokenized and normalized

TF-IDF:

* <https://radimrehurek.com/gensim/auto_examples/tutorials/run_scm.html#sphx-glr-auto-examples-tutorials-run-scm-py>
* Function: genism.models.TfidfModel

Doc2Vec:

* <https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html#sphx-glr-auto-examples-tutorials-run-doc2vec-lee-py>
* Represents documents as vectors
* Better than averaging Word2Vec vectors
* Doc-vector:
  + Distributed Memory (PV-DM)
    - analogous to Word2Vec CBOW
    - doc-vector is obtained by prediction of centre word = average of context word-vectors and full document’s doc-vector
  + Distributed Bag of Words (PV-DBOW)
    - analogous to Word2Vec SG
    - doc-vector is obtained by prediction of target word from the full document’s doc-vector
* <https://pub.towardsai.net/an-intuitive-introduction-of-document-vector-doc2vec-42c6205ca5a2>

Universal Embedding:

* <https://medium.com/analytics-vidhya/sentence-embeddings-facebooks-infersent-6ac4a9fc2001> (15.08.2023)
* embeddings that are pre-trained on a large corpus and can be plugged in a variety of downstream task models
* uses general word/sentence representations learned on the larger dataset

Infersent:

* <https://medium.com/analytics-vidhya/sentence-embeddings-facebooks-infersent-6ac4a9fc2001>
* Universal Embedding
* Architecture
  1. Sentence encoder: encodes word vectors to sentence vectors
     + BiLSTM with max/mean pooling
     + LSTM and GRU
     + Self attentive network
  2. NLI classifier: takes encodings and returns class (neutral, entailment, contradiction)

Databases:

* <https://www.astera.com/de/type/blog/a-quick-overview-of-different-types-of-databases/>
* Relational Database (RDBMS):
  + Each entry must have certain schema.
  + ACID= atomic, consistent, isolation, durability
* Non-relational database:
  + No schema.
  + Key-value: e.g., Amazon DynamoDB, Redis
  + Document-based: data saved as JSON, similar structure. E.g., MongoDB, Couchbase
  + Search Engines: enables text-based search for data, saves documents in JSON format (https://de.wikipedia.org/wiki/Elasticsearch), e.g., Elasticsearch

Elasticsearch (Database):

* <https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html>
  + Fast search for different datatypes
* <https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-overview.html>
  + Offers text search (incl. automatic tokenizing & normalizing) on (type=) text fields
* Search: KNN: <https://www.elastic.co/guide/en/elasticsearch/reference/current/knn-search.html> and <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-search.html#search-api-knn>
* Init: <https://github.com/fujaba/fulib.org/blob/develop/docker-compose.assignments.yml#L18>
  + Image in Docker: <https://hub.docker.com/layers/library/elasticsearch/8.8.1/images/sha256-c6268f420d8d15de1e5212b322b86acabf1690640920d26ed405db96b68eb673?context=explore>
    - 8.8.1 is layer combination id, which identifies data files that compose elasticsearch
  + Environment: one computer = single node
  + Port: 9200 of local host is mapped on 9200 virtually
  + Volumes: save upper layer data from running command in certain volume (persistent)

Stemmer

* <https://www.elastic.co/guide/en/elasticsearch/reference/current/stemming.html>
* Algorithmic stemmer
  + Stem words based on rules.
  + Typically, faster
  + Use little memory
  + Work well & little setup
  + Do not work well with irregular words not containing their root form (be etc.)
  + E.g., stemmer, kstem, porter\_stemmer, snowball
* Dictionary stemmer
  + Stem words by looking them up in dictionary.
  + Good for irregular words
  + Good for similar spelled, but different words
  + Worse performance
  + Stemmer only as good as dictionary
    - Incomplete, outdated
  + A lot of RAM usage
  + E.g., hunspell

Tokenizer

* <https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenizers.html>
* Word oriented tokenizers
* Partial word tokenizers
* Structured word tokenizers