Bachelorarbeit – Methoden

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.

Latent Dirichlet Allocation (LDA):

* <https://towardsdatascience.com/end-to-end-topic-modeling-in-python-latent-dirichlet-allocation-lda-35ce4ed6b3e0>

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

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

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