INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

ART2VEC

a semantic search engine for tagged artworks based on word embeddings

Jaderson Rafael Webler

Bachelorarbeit

Aufgabensteller	Prof. Dr. François Bry
Betreuer	Prof. Dr. François Bry, M. Sc. Martin Bogner
Abgabe am	20.02.2019

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 20.02.2019 Jaderson Rafael Webler

Abstract

This bachelor thesis reports on the implementation of a semantic search engine based on word embeddings. The dataset comes from the ARTigo ecosystems, which comprises at the same time a tagging system (folksonomy) and a search engine for images of artworks. Word/document embeddings consist of a technique for representing a textual document as a vector of real numbers. The basic premise of word embeddings is that words or documents which appear in similar contexts are represented by similar vectors. In the course of this thesis, the two word embediding techniques word2vec and doc2vec were both trained and evaluated in order to creating a semantic artwork search engine for the ARTigo dataset, namely art2vec.

Many word embedding techniques rely on deep learning models based on neural networks. Almost all of word2vec's hyper-parameters were experimented and refined with the focus on improving the recall for search queries, i.e. finding artworks not explicitly tagged with the queried search terms. Word2vec and doc2vec include two different algorithms for training the embeddings, which are differently suited for dissimilar tasks.

Both a vector representation and a simpler synonym list generation approach were experimented. Considering various semantic aspects, the quality of the semantic similarity between words and documents that are extracted from the embeddings were evaluated. So that these document vectors can be integrated into a search engine, an algorithm for generating query vectors from ARTigo's search queries is presented in this thesis.

Finally, an evaluation by users is reported about the recall improvements resulted from each of the applied word embedding approaches. Word2vec's averaged word vectors outperformed the other two approaches in all tag categories evaluated. Indeed good results were achieved only for surface (67.90%) and deep semantics (77.16%) tags. So that specially long queries can offer a good retrieval quality, in future work word2vec parameters have to be refined again and the developed query vector algorithm must be polished.

Zusammenfassung

Diese Bachelorarbeit berichtet die Konzeption einer semantischen Suchmaschine basierend auf "Word Embeddings" (Wort-Einbettung). Der Datensatz stammt aus einem System namens ARTigo, das gleichzeitig ein Tagging-System (Folksonomy) und eine Suchmaschine für Bilder von Kunstwerken darstellt. Das Einbetten von Wörtern/Dokumenten besteht aus einer Technik, bei der ein Textdokument in einem Vektor reeller Zahlen dargestellt wird. Die Grundannahme für das Einbetten von Wörtern ist, dass Wörter oder Dokumente, die in ähnlichen Kontexten erscheinen, durch ähnliche Vektoren repräsentiert werden. Im Rahmen dieser Arbeit wurden die beiden Worteinbettungstechniken word2vec und doc2vec sowohl trainiert als auch ausgewertet, um eine semantische Artwork-Suchmaschine für den ARTigo-Datensatz, nämlich art2vec, zu erstellen.

Viele Techniken zum Einbetten von Wörtern gründen auf Deep-Learning-Modellen, die auf neuronalen Netze basiert sind. Nahezu für alle Hyperparameter von word2vec wurden Experimente durchgeführt, in denen die Parameterwerte verfeinert wurden. Der Fokus dabei lag darauf, den Rückruf (Recall) für Suchanfragen zu verbessern, d.h. das Finden von Kunstwerken, die nicht explizit mit den abgefragten Suchbegriffen gekennzeichnet sind, zu ermöglichen.

Experimente wurden sowohl für eine Vektordarstellung als auch für einen einfacheren Ansatz zur Erzeugung von Synonymlisten ausgeführt. Unter Berücksichtigung verschiedener semantischer Aspekte wird die Qualität der semantischen Ähnlichkeit zwischen Wörtern und Dokumenten, die aus den Worteinbettungen extrahiert werden, bewertet. Um diese Dokumentvektoren in eine Suchmaschine integrieren zu können, wird in dieser Arbeit ein Algorithmus zur Generierung von Anfragevektoren aus ARTigo-Suchanfragen vorgestellt.

Schließlich werden bei einer nutzerbasierten Auswertung Recall-Verbesserungen in den Suchergebnissen, die durch jeden der angewandten Word-Embedding-Ansätze hervorgerufen wurden, beleuchtet. Die durchschnittlichen Wortvektoren von word2vec übertrafen die beiden anderen Ansätze in allen bewerteten Kategorien. Tatsächlich wurden nur für die oberflächlichen semantischen Tags (67,90%) und die vertieften semantischen Tags (77,16%) gute Ergebnisse erzielt. Damit lange Suchanfragen auch eine gute Abrufqualität bieten können, müssen in Zukunft die word2vec-Parameter erneut verfeinert und der entwickelte Abfragevektoralgorithmus optimiert werden.

Acknowledgments

First, I would like to thank Prof. Dr. François Bry for his supervision which contributed to make this bachelor thesis possible. Likewise, the coordination of Martin Bogner, M. Sc., was decisive for the quality of the presented research. Also, I would like thank the NLP community in general for providing inspiration through their discussions. Finally, I have to thank all the users that participated in the evaluation of this work.

Contents

CHAPTER 1

Introduction

In information retrieval (IR), a search engine is a system designed to find specific information on a computer or in a computer network [9]. This system aims to minimize the time required to find information and the amount of information which must be consulted. Search engines can be assigned to many classes: source (file system, web, human computation), content type (full text, image, video), interface (incremental, semantic search), topic.

This bachelor thesis is limited to search engines for documents that are tagged (annotated) via human-based computation games, or often called games with a purpose (GWAPs). Such kind of search engine was created by a system called ARTigo [26, 25], which offers several GWAPs for annotating images of artworks. The collection of artwork pictures available on ARTigo is mostly authored by European artists from "the long nineteenth century", an artistic and linguistic period starting at the end of the 18th century and ending at the beginning of the 20th century.

The implementation in this thesis is based on the data acquired by ARTigo for building an image search engine. Image retrieval systems are computer systems for searching digital images mostly from large databases. Though great progress within applications on computer vision has been made in recent years, the most common image search systems are based on metadata such as keywords/tags.

Just as in full text search engines, these keywords are indexed and stored in a database, and then used to match search queries whose results (documents) are ranked by relevance. The usefulness of a search engine depends on the relevance of the results it returns. In order to find out what the most relevant documents are for a particular search query, there are many aspects that need to be observed within an IR system. The classic IR models such as vector space (VSM) and probabilistic models do not overcome the many challenging problems on document relevance ranking. VSM relies on vector representation of documents and queries mostly based on term frequency.

One of the most challenging problems in IR systems founded on natural languages is detecting the semantics of documents and queries. In general, different words can have the same meaning (synonymy) while one word can have different meanings (polysemy). Consequently, many relevant documents are not retrieved by a regular search engine that matches only across keywords. The issues caused by this linguistic phenomenon can be reduced by considering contextual related words or synonyms in the ranking process. Using concepts of semantic search and applying techniques of natural language processing

(NLP) is an efficient way of measuring the semantic similarity along documents. NLP is a subfield of computational linguistics and artificial intelligence which focuses on algorithms to process and analyze large amounts of natural language data.

Processing very big quantity of data in trivial time has turned into a common computational task in the last few decades. Hence, many subfields of computer science have re-emerged including NLP. Therefore, popular web search engines such as Google and Bing have become quite refined. Open source platforms for implementing search engines have also profited from the progresses on computational time cost. These days, APIs such as Apache Solr[1] and Elasticsearch[2] have become the consolidated state-of-the-art tools that help build search engines.

Google's most recent search infrastructure makes use of NLP for analyzing the entered keywords and considering the analysis results in the ranking of documents along the internet. For example, their synonym system helps the algorithm know what the user means, even if a word has multiple definitions[3]. Under many NLP techniques for discovering relationships among words, one that has recently grown in attention is the use of neural networks for generating word embeddings. Word embeddings consist on mapping the words of the vocabulary into vectors of real numbers. This roots back to the vector space model in IR. Word embeddings can be seen as a technique for weighting terms in a term-document matrix of predefined and fixed number of dimensions. In short terms, words and documents that are similar tend to have similar vector representations. Word2vec [10, 11], GloVe [15] and fastText [7] are competitive word embedding approaches in trend at the moment.

Motivation

This thesis reports the implementation of a semantic search engine based on tagged images of artworks. The data source comes from the previously mentioned system ARTigo, in which users can tag images by playing several games with a purpose (GWAPs). Up to November 2018, 68.891 artworks were tagged summing 9.235.864 taggings in German, 997.134 in English and 44.456 in French. ARTigo offers with no doubt an efficient collaborative tagging system (so called folksonomy) for artworks through its many different GWAPs. Nevertheless, there is still a research gap between the collected taggings and the use of them to implement a search engine.

A search engine such as ARTigo faces many other issues besides polysemy and synonymy that are deciding for an accurate image retrieval. For example, spelling errors can occur very often when users enter text as input. A spelling error is characterized by the grammatically wrong form of the entered word. Differing from spelling errors, when a user accidentally inputs a wrong letter that is nearby within the keyboard, or one letter just ends up missing, this is called typo. In ARTigo's platform, typos and spelling errors are good examples of problems that can appear in both queries and documents, since the text content of the documents are also generated by regular users and cannot be validated/corrected without a lot of human effort.

Automatically detecting and correcting spelling errors and typos require the use of several NLP techniques. Therefore, the work in this thesis will concentrate on scaling semantic similarity between documents with the use of word embedding frameworks to retrieve images of artworks. However, the promising results that can be gained with word embeddings cannot be easily integrated into a search engine. This is due to the fact that a search query reside in a different space than documents. Still search queries must have the same

[1]Apache Solr: `http://lucene.apache.org/solr/`

[2]Elasticsearch API: `https://www.elastic.co/`

[3]Google's search process: `https://www.google.com/intl/ALL/search/howsearchworks/algorithms/`

dimension as documents, so that they can be compared against each other. A word embedding approach can just imply a vector representation for terms present in the given search query, instead of semantically representing these terms as a query related to expected similar documents. That imposes a challenge when using word embedding techniques such as word2vec in search engines.

Another issue of same relevance is that the efficiency of the word similarities can vary a lot by type of text corpora and strongly depends on the chosen algorithms along with their several possible parametrizations. To minimize the spectrum of these interdependent problems, many different integration methods can be applied as well as automated evaluation tests for word embedding models are available.

In chapter 3, two different approaches are presented. The main apporach is divided into two sub-approaches. First, word2vec is trained on ARTigo's taggings and the generated word vectors are used to build groups of related words that are written into Solr's synonym lists. Thus, for the most frequent terms in the vocabulary also their related terms within the dataset are taken into consideration by the search process. The second approach within word2vec relies on averaging the trained word vectors into document vectors in order to compare them by cosine similarity. After refining the parameters through automated evaluation, both of these approaches can retain the semantic meaning between the documents. Therefore, this technique is very likely to improve the recall, since many relevant artworks containing none of the query terms might be included in the result set.

The second approach within word embeddings is to train a doc2vec [8] model (an extension of word2vec) which represents not only words but also documents as vectors. These document vectors are extracted and imported into Solr's core. Every search query is initially given as full text to the previously trained doc2vec model which estimates the vector representation for that text. In the end, this query vector is used by a Solr plugin to build a vector scoring based on cosine similarity or dot product. Depending on the size of the document collection and the number of vector dimensions, the vector scoring can consume an undesirable amount of time comparing to other scoring approaches.

Ranking algorithms are one of the important factors in the basis of a search engine. The most used IR model is the vector space model (VSM), which achieves better performance for general document collections compared to boolean and probabilistic models. In VSM, documents and queries are represented by vectors of same dimensions and each dimension corresponds to an individual term. If a term occurs in the document, its value in the vector is non-zero. These term weights are often computed by a statistical method called "term frequency"-"inverse document frequency" (TF-IDF) that is intended to reflect how important a word is to a document in a corpus. ARTigo's current search is based on the search engine platform Apache Lucene/Solr, which implements the VSM model. This work keeps Solr as underlying infrastructure with the goal of making a future integration with the current search on the production website much easier. Besides that, some of the developed applications could be used to build or extend a Solr-based search engine with semantic features for text data of any topic.

CHAPTER 2

Related work

Image search engines can differ in several aspects. One of them is the original source of the texts in natural language that represent the documents. So that images can be retrieved without using computer vision, their metadata and/or descriptions have to be first provided. The image description is often composed by user-generated tags. In the end, the source text (tags) represent a document in IR, which is assigned to an image file. ARTigo is a platform where users alone produce the source text by tagging artwork images. The ARTigo search engine takes these same texts to provide users search results that match queries entered also by other users that might not have contributed to collect the data. ARTigo's tagging system is built up on "games with a purpose" (GWAPs), in which the tags are collected in a recreational way by playing games. These human-generated tags result in a data structure called folksonomy [24]. This data structure is defined by a tuple (U,T,R,Y) of sets U,T,R and a tripartite relation $Y \subseteq U \times T \times R$. These sets are composed by users U, tags T and resources R (images in this case), while Y represents the set of tag assignments. A *post* is the collection of tag assignments with the same user and same resource.

Currently, ARTigo offers a simple keyword search and a advanced search by artist, title, year and location. The result images are in a list with their tag clouds. As IR model, ARTigo's search engine implements VSM with TF-IDF [18] as weighting factor which is a good technique for building general search engines. TF stands for term frequency, representing the number of occurrences of a particular term in a particular document. IDF (inverse document frequency) corresponds to the number of documents in the collection to which the term is assigned.

Given a set of documents D, a document d and a term t, TF-IDF(t,d,D) is defined by the product of both measures

$$tf(t,d) \cdot idf(t,D)$$

Depending on the use case, different weighting schemes for TF and IDF can be applied. In the VSM model, a document d and a query q are represented by vectors of weights (TF-IDF) of same dimension that are defined as $d_j = \{w_{1_j}, w_{2_j}, \ldots, w_{n_j}\}$ and $q = \{w_{1_q}, w_{2_q}, \ldots, w_{n_q}\}$, where w_i is the weight for the term i. In relevance ranking, the similarity between a query and a document is often calculated by the cosine of the angle between their vectors

$$cos\theta = (d \cdot q) \ / \ (||d|| \ ||q||)$$

where $||d||$ is the norm of vector d and $||q||$ is the norm of vector q.

Although VSM with TF-IDF delivers good retrieval quality in common use cases, this model has a number of shortcomings. One of the main issues is that in VSM terms are seen as unrelated entities, thus the semantics of documents and queries are not captured. Also, the dimension of the term-document matrix grows proportionally to the number of terms in the index. Furthermore, the document representation cannot be easily extended to other structures besides terms. If one would like to represent a document with extra features such as topic or subtopic and use these features while ranking the documents, then TF-IDF weighting has to be at least adapted for each new feature to function properly.

In order to overcome these classic VSM problems, recent word embedding techniques can be used. Word embeddings are representations of words in a low-dimensional vector space. Such concept was first introduced by Deerwester et al in the work "Indexing by latent semantic analysis". While some word embedding techniques work on factorizing a term-document matrix, other train a model to predict a word from a given context, such as word2vec. There are other word embedding methods based on graph algorithms such as LINE [22] and DeepWalk [16]. An earlier approach for representing words as high-dimensional vectors is presented by Schütze and Pedersen [21], in which words are defined to be similar if they have similar co-occurrence patterns. However, this technique has shown to deliver lower correlations with human intuition comparing to some of the other mentioned methods [2].

2.1 Latent semantic analysis (LSA)

A search engine for ARTigo that applies word embeddings was already conceptualized by Wieser et al. [26, 25] using higher-order latent semantic analysis (LSA). LSA, also known as latent semantic indexing (LSI), is an indexing technique that uses a method from the linear algebra called singular value decomposition (SVD). Higher-order means that the data structure has an order greater than matrices. SVD consists in the factorization of a real or complex matrix (i.e. tensors) and is used to recognize relationships between words in unstructured text. It projects each word in a subspace of a predefined number of dimensions. As soon as the vectorial representation of words is obtained, the semantic similarity between two term vectors can typically be measured by the cosine of the angle between their respective vectors.

In Wieser's work, the proposed semantic search also models the users that collected the tags of each artwork. For that reason, a tensor instead of a matrix is needed to create the mathematical structure for the model. Just like a matrix can be seen as a multidimensional vector, a tensor can be seen as a multidimensional matrix. After applying SVD on them, these tensors are transformed back into matrices so that the cosine similarity can be computed. Higher-Order SVD can help to overcome synonymy (multiple words with similar meaning) while it lacks on capturing polysemy (multiple meanings of a word). The computational costs of this method are very high but its runtime complexity can be reduced via parallelization [25]. However, this implementation is currently not deployed in the ARTigo's production website.

2.2 Word2vec

After LSA being popularized in the early 1990s, word embeddings have gained attention again when word2vec was released in 2013. Word2vec is one of the word embedding approaches that use neural networks for producing the vector representation of words. In the modern days, an artificial neural network is mostly structured by three layers (input, hidden, output) containing elementary units called artificial neurons or nodes. Each node receives one or more inputs that are often weighted, and then the sum of the inputs are passed into a non-linear function known as activation function. The activation function defines the output of a node given an input or set of inputs. It also maps the resulting values into desired ranges such as between 0 to 1 or -1 to 1. Its output is then used as input for the next node and so on, until a desired solution to the original problem is found.

A subfield of artificial intelligence, machine learning (ML), aims to effectively perform a specific computational task with use of models and inference instead of giving explicit instructions. These models rely on algorithms that are used to predict the solution for a problem based on a set of examples of problems (data entities) often together with their respective solutions (labels). This set of sample data is also known as training data. The most important difference between applications based on machine learning techniques and regular programming is that in ML the system needs to know *what* has to be solved and not explicitly *how* to solve the problem. Many ML algorithms use neural networks to train data in order to make predictions or decisions without the need of custom programming for that specific topic. Most of ML techniques apply feature extraction[1], which is a dimensionality reduction process, where an initial set of raw variables is reduced to more manageable groups (features) for processing, while still accurately and completely describing the original data set. This process results in a feature vector, what can be comparable to the word vectors generated by word2vec.

Basically, word2vec is used for finding semantic similarity among words. A list (corpus) of lists of words (documents) is given as input, and the output is a list of vectors of same dimension, each containing values between -2 and 2 for every dimension. Each of these vectors represent one unique word present in the input list. Two or more vectors can then easily be compared via cosine similarity, whose result defines the semantic relatedness between the words assigned to these vectors. Word2vec has a set of hyper-parameters that can be refined to train different kind of datasets.

Word2vec is a set of two independent shallow, two-layered neural networks: continuous bag of words (CBOW) and skip-gram. In these models, a window of a given size moves along the corpus, and in every step the language model is trained on the words within this context span. In essence, the CBOW algorithm predicts a word based on a given context while the skip-gram predicts a context based on a given word [10].

2.2.1 CBOW model

CBOW is similar to the feedforward neural network language model (NNLM), which consists of input, projection, hidden and output layers. On NNLM, the N previous words are encoded in the input layer using *1-of-V* coding, where V is size of the vocabulary. The bag of words model removes the non-linear hidden layer from NNLM and shares the projection layer with all words (not only the projection matrix). Since the word vectors are averaged, the word position in the vocabulary does not influence the projection. The model is called continuous BOW because it takes words from both history and future context [10].

2.2.2 Skip-gram model

Skip-gram is similar to CBOW, but instead of predicting the current word based on the context, it tries to maximize the classification of a word with base on another words in the same sentence. More specifically, it uses each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word. Increasing the context range (window) improves quality of the resulting embeddings, though it also leverages the computational complexity. Skip-gram

[1]DeepAI's glossary: `https://deepai.org/machine-learning-glossary-and-terms/feature-extraction`

gives less weight to the distant words by sampling less from those words in the training examples, seen that the more distant words are usually less related to the current word than those close to it [10].

2.2.3 Optimization strategies

Both CBOW and skip-gram architectures are equipped with two distinct optimization techniques: hierarchical softmax (HS) and negative sampling (NEG). The key benefit of HS is that, to obtain the probability distribution, only about $log_2(W)$ nodes need to be evaluated in the neural network, instead of evaluating W output nodes. Hierarchical softmax represents the output layer with the W words as leaves of a binary tree and for every node, there is a representation of the relative probabilities of its child nodes. These features determine a random walk that assigns probabilities to words. In word2vec, a binary Huffman tree is used for HS, what makes a significant difference on the performance, as it assigns short codes to the frequent words what results in faster training [11].

Noise contrastive estimation (NCE) is a technique which supposes that a good model should be able to differentiate data from noise through logistic regression, being an alternative to the hierarchical softmax approach. Negative sampling simplifies NCE. While NCE needs both samples and the numerical probabilities of the noise distribution, NEG uses just samples. Noisy data defines the additional meaningless information in it called noise. Hence, it distinguishes the target word w_0 from draws of the noise distribution $P_n(w)$ using logistic regression, where there are k negative samples for each data sample. NEG shows not only to be faster, but it also outperforms hierarchical softmax in the tests performed in [10] for the Google News dataset.

2.2.4 Word vectors evaluation

According to Mikolov's experiments [10], skip-gram performs better on semantic and CBOW stronger on syntactic tasks. For example, skip-gram tends to outperform when one makes analogy questions such as "does *man* relates to *woman* as *father* relates to *mother*?" because, unlike CBOW, it considers the word order in a phrase. CBOW however seems to detect syntactic features such as plural-singular relatedness (*men* relates to *man*) better than skip-gram. These are the results of experiments with a big and general corpus in the English language. That means for every type and amount of data these both models could and should be used differently than in the mentioned evaluation.

In order to evaluate the accuracy of the trained word vectors in all parameter experiments, a test set of analogical reasoning for semantic and syntactic tasks that contains both words and phrases has to be developed. A typical analogy question pair could be "Montreal":"Montreal Canadiens"::"Toronto":"Toronto Maple Leafs". It is considered to have been answered correctly if the nearest representation to vec("Montreal Canadiens") - vec("Montreal") + vec("Toronto") is vec("Toronto Maple Leafs"). Similar to that, a simple vector addition can often produce results comparable to vec("Germany") + vec("capital") is close to vec("Berlin") [10]. The answer to one such analogy question obviously depends on the trained vocabulary and the context of the entire corpus. The more general and big the corpus is, the more accurate word2vec gets.

Word2vec includes a test set for evaluating models trained on corpora in the English language. Evaluation test sets for the German language have been developed by Müller [13] based on word2vec standards. These test sets were built to evaluate a trained Wikipedia corpus. Although the results of a particular semantically strong model were good with this test set, it should not be simply applied the same way for the evaluation of models trained on any kind of data. If the vocabulary in the corpus does not cover the one of the test set, then the quality of the evaluation reduces.

2.3 Doc2vec

The technique of learning semantic meanings among words using neural networks has been applied also at the document level [7]. While word2vec is able to build numerical representations (word vectors) of words and phrases, it cannot accomplish the task of representing documents as numerical vectors. Words put together build mostly logical structures based on a set of rules (syntax). In contrast, documents within whichever collection do not have any logical ordering. To address this issue, doc2vec has been implemented.

With raw word vectors, one is not able to find similarities between documents that contain the respective words. A common workaround is to average these word vectors to build document vectors. During averaging, a lot of semantic information from the word vectors can be lost. Doc2vec delivers document vectors that include the semantic similarity between them. That means, they can be compared with each other the same way word vectors can be compared in word2vec.

With doc2vec one is able to generate paragraph (sentence) embeddings of predefined dimensions along with word embeddings. This is basically done by adding the documents' IDs into the training data. Doc2vec is heavily based on word2vec and it also consists in two main algorithms: distributed memory version of paragraph vector (PV-DM) and distributed bag of words version of paragraph vector (PV-DBOW). Syntactically seen, an information entity can usually start with a word, then it grows into phrases, which construct sentences and these are grouped into paragraphs. A document is nothing more than a set of paragraphs. In a nutshell, word2vec is capable of generating word and phrase embeddings, while doc2vec can process from sentence to document embeddings.

PV-DM is a small extension of word2vec's CBOW model while PV-DBOW has a similar structure to word2vec's skip-gram model. The memory usage in the PV-DBOW model is much lower for the reason that its architecture does not require to store the word vectors. In contrast, the PV-DM model shows in general to perform better results. Since there is too little research on doc2vec with folksonomies, a comparison with both training algorithms on ARTigo's data would provide a good research opportunity.

2.4 Word embeddings integration with search engines

The goal of this section is to provide insights on how to integrate word embeddings with ARTigo's existing infrastructure, namely Apache Lucene/Solr.

2.4.1 Synonym lists

The simplest way to use word2vec's results in Solr is building a file that contains lists with the words associated to their related words or synonyms. Solr offers a built-in synonym graph filter[2] which maps single- or multi-token synonyms, producing a fully correct graph output. This filter can be easily configured to parse search queries and it can also be used for indexing multiword expressions. A great set of tools that accomplishes these features for the English language was developed by Dice[3]. It includes weighted synonym term expansion at query time.

Solr's synonym file must contain one word per line with its list of related terms. One can map synonyms in two different kinds of lists: single comma-separated lists of words (e.g. *couch,sofa,divan*) and pair of comma-separated lists of words with the symbols "=>" between them (e.g. *ginormous,humungous* => *large,huge*). With the option, if the token matches

[2]Apache Solr's developer guide: `https://lucene.apache.org/solr/guide/6_6/filter-descriptions.html`
[3]Dice's plugins: `https://github.com/DiceTechJobs/SolrPlugins/`

any of the words, then all the words in the list are substituted, which will include the original token. In the other way, if the token matches any word on the left, then the list on the right is substituted and the original token will not be included unless it is also in the list on the right. Nevertheless, depending on the structure of the synonym file and how the synonym graph filter was applied, the quality of the search results can vary a lot.

2.4.2 Vector approaches

Word2vec delivers different information about the word embeddings it builds. One of them is the vector representation for each word. A document could be seen as the sum or the average of the vectors of all its words. This approach is also known as simple averaging of word vectors. Another algorithms such as Google's Universal Sentence Encoder [3] can be used to build the so called sentence/paragraph/document vectors. It consists on a deep averaging network (DAN) where input word embeddings and bigrams are averaged together and passed through a feed-forward deep neural network.

Applying this for a Solr-based search engine, one could store these vectors as documents on Solr's index and use them for ranking. In order to do that, the queries need to be transformed into vectors as well. Unfortunately, word2vec does not offer a built-in method that translates a list of words into vectors based on the context of the loaded model. This issue is addressed by doc2vec. Doc2vec computes a vector representation for a particular new document in the corpus. This new document can be seen as the set of words contained in the search query. Since similar documents are represented by similar vectors, the query vector gets a similar representation to the documents related to it.

In doc2vec, a vector tries to grasp the semantic meaning of all the words in the context by placing the vector itself in each context. Hence, a document vector contains the semantic meaning of all the words in trained the corpus. In word2vec, each word preserves its own semantic meaning. Thus, summing up all the vectors or averaging them will result in a vector which could also have all the semantics preserved. If the vectors for *transport* and *water* are added, the result is almost equivalent to *ship* or *boat*. That means, the sum of the vectors leverages the semantic representation of the sentences by which they are composed. This approach could be used alternatively to doc2vec also for generating document and query vectors.

The last factor on vector scoring is to discuss the actual scoring process. Once the query vector is obtained from doc2vec or word2vec, Solr has to calculate the cosine similarity between the query and the document vectors. This technique can express better semantic relationships between queries and results than TF-IDF scoring. However, the quality of the results as well as the computational cost of the scoring process strongly depends on the dimensionality of the word vectors trained. Vector query parsing and vector scoring plugins for Solr were recently implemented [[4] [5]].

An alternative to vector scoring plugins or Solr's synonym filter could the library Deeplearning4J[6], which implements several neural network models including an out-of-the-box version of word2vec for the JVM. Apache Solr/Lucene is Java-based, so this lib could be used to load (or train too) a word2vec model directly inside an existing search engine. However, there is in advance no token filter developed for integrating Deeplearning4J's word2vec with Lucene's search process. Nor is there an integrated scoring function that work on vectors of predefined size. Thus, such a plugin must first be designed so that word2vec can work with Solr without using static synonym files or any other call stack for retrieving query vectors. With such solution, a good integration of the ML results in to the existing search infrastructure could be automated. That way, search indexes along with the

[4]Solr vector scoring plugin by Almodarresi: `https://github.com/saaay71/solr-vector-scoring`
[5]Solr vector scoring plugin by Dice: `https://github.com/DiceTechJobs/SolrPlugins/`
[6]Deeplearning4J: `https://deeplearning4j.org`

constant grow of word relatedness in the dynamic corpus would never get obsolete.

2.5 Comparative studies on word embedding algorithms

The high number of parameters in word2vec requires a deeper investigation of their configuration for every kind and size of text corpora. The vector size determines the number of dimensions of the produced embeddings. Higher embedding dimensions can store more information, and therefore permitting higher levels of expressiveness for the vector. Another relevant parameter is the initial learning rate of the algorithm, which defines its convergence speed. Fine-tuning that parameter is crucial to receive optimal results, because if chosen improperly, the learning process either converges very slowly or might be unable to converge at all [14].

Regardless of LSA being one of the most used methods for analyzing semantic relations of words in documents, models based on neural networks have recently gained attention in this field. Although it is not very clear if these models outperform LSA [2], word2vec has turned into a popular word embedding technique. The main difference between word2vec and LSA is that word2vec is a prediction-based model while LSA is a counter-based model. As word2vec's memory usage is lower compared to LSA, this can be seen as an advantage for processing large datasets. On the other hand, word2vec's performance drastically reduces if being trained on smaller sets of data [1]. For that reason, word2vec is only applicable for datasets with approximately 10 million words, what almost perfectly accords with the current amount of ARTigo's taggings in German ($>$ 9 million).

Testing another word embedding methods such as LSA along word2vec with different datasets is a common activity in this field. Checking out the range of the parameter values used in [1] can be helpful to estimate the ones likely to fit for the dataset to be experimented with. In the referenced paper, a case study of LSA and word2vec is made for small corpora. Word2vec tends to produce more refined word embeddings than LSA if the corpus size exceeds a minimum threshold. However, the conducted experiments used only skip-gram and negative sampling, as well as no minimum word frequency and all other parameters with their default values [1].

A meaningful comparative method between word2vec, GloVe and LINE including parametrization experiments is presented in [14]. This work is seen as specially relevant for the current thesis, because their tests are executed on two categories of datasets for evaluating embedding algorithms on tagging data. In each category, experiments are accomplished with three different datasets from tagged data of several topics. Although they test word2vec only with the CBOW architecture, the range of values for the evaluated parameters is wider than in [1].

In a quality evaluation, the work in [12] presents a few of the chosen techniques for this bachelor thesis. Common metrics such as MAP and MP@3 were used for optimization and evaluation. They evaluate the results with two test datasets: one small and one medium sized. Their technique of combining averaged word vectors from word2vec with TF-IDF is the best to perform in the two tested datasets. With doc2vec, there has been some computational problems that lead to limitations on its training process and thus ending up as a failed approach. They also apply the use of synonyms but with the goal of leveraging the quality of search results returned directly by a word embedding model. That is similar to what one can do with synonyms generated by word embeddings to improve Solr's TF-IDF scoring. However, the synonym attempt in their study has brought no relevant results. Despite the good quality of the word embeddings, words have more than one meaning (polysemy), and word embeddings are not able to distinguish to which meaning a specific word in a query is referring to, thus leading to valid synonyms that do not relate to the query [12].

CHAPTER 3

Concepts behind art2vec

In this chapter, the methodology of the implementation of a semantic search engine called art2vec is presented. First, a deeper look into the dataset and its pre-processing is taken. The pre-processed data is then given as input into different training algorithms of word and document embedding techniques: word2vec and doc2vec. In this section, all used models are illustrated and their main parameters are listed. The training processes of the applied algorithms in word2vec and doc2vec are briefingly described. Doc2vec is capable of inferring document vectors from a pre-trained model based on raw text input. For word2vec, a technique of averaging word vectors into document vectors is used. At the end of this chapter, an algorithm for generating query vectors from search requests is proposed and the proceedings for integrating its results into Solr are explained.

3.1 Pre-processing

Word2vec was built to work with sequential data in as much as the word's position in the sentence is taken into consideration for generating the word embeddings. However, the sequence of tags normally does not hold any meaning [14], regarding the concept of ARTigo's tagging system itself. In fact, if the sequence of the tags is changed, then word2vec will produce different word embeddings. For that reason, this bachelor thesis takes the challenge of experimenting with several possible configurations of the pre-processed tags (see Table 3.1). As doc2vec is based on word2vec, it is reasonable to take the configuration that achieves the best results with word2vec and then use it for training doc2vec. Taking different pre-processed data to experiment with doc2vec can be seen as future work.

In the folksonomy aspect, in contrast to Wieser's work [25], the current thesis does not consider the set of users that tagged the images. Structuring the artworks by users who assigned their tags would not be a suitable document representation to be passed into word2vec's or doc2vec's algorithms. Therefore, the folksonomy structure needs to be optimized to $(T, R, Y) \mid Y \subseteq U \times T \times R$. Where T is the set of tags, R is the set of resources (images of artworks) and Y the set of tag assignments. Based on the proceedings in Niebler et al. [14], the tag assignments have been grouped into sentences with varied tag position, as described further in this section.

Depending on the dataset, pre-processing raw text in several different ways can be very time-consuming. Therefore, this project has developed a small tool for the automatic gen-

several representations for the same document per parameter configuration				
tag_c	*c_max*	*outer_s*	*inner_s*	result document
0	0	0	0	*woman red sky*
0	0	1	0	*red sky woman*
1	0	0	0	*woman woman woman red sky sky sky*
1	0	1	0	*sky sky woman woman woman red*
1	0	1	1	*sky woman red woman sky woman*
1	2	1	1	*sky woman red woman sky*

Table 3.1: samples of different possible pre-processed versions for a document containing the set of tags *tag(tag_count)*: *woman(3) red(1) sky(2)*

eration of different tag pre-processing structures by refining several parameters:

tag_cooccurrence if 1, for every tag, repeat it based on tag count;

cooccurrence_max restricts the co-occurrence of tags until a maximum given, 0 for co-occurrences based on tag count;

outer_shuffle if 1, randomize the order of groups with and without *tag_ cooccurrence*;

inner_shuffle if 1, this option applies if *outer_shuffle* is 1 as well. If *tag_cooccurrence* is also 1, then the *inner_shuffle* has the effect of completely randomizing the sequence of the tags including their co-occurrence, if not, then it just makes a double shuffling.

On ARTigo's platform, an artwork is represented by its unique ID, title, artist name, date, location and tag assignments. A tag assignment for each artwork is composed by name, language and number of times the tag was assigned to the artwork (tag count). In the pre-processing stage, only the artwork ID and its taggings are relevant and all the rest of the data is used later for auxiliary purposes. This data can be requested to ARTigo's RESTful API at any time.

In ARTigo's case, all the tags/keywords are stored lowercase, what makes part-of-speech (POS) tagging for word-sense disambiguation in German almost unfeasible. POS-tagging consists in labeling which syntactic role a word has in its sentence. Usual parts of speech are NOUN, VERB, ADJ, but there are also more fine-grained ones like NP or NC (proper or common noun). A technique called name entity recognition (NER) aims to address such problem of classification of nouns, though this NLP task presents several issues specially for the German language. An approach for classifying German name entities including recursive recognition of compound nouns is proposed in [5]. In the end, POS-tagging could disambiguate many homonyms present in the dataset such as *Weg* (path) and *weg* (gone), *Fest* (party/festival) and *fest* (firmly/fixed).

Word2vec and doc2vec both take a list of lists of tokens as data input. However, the data source file must be converted into a format suitable for word2vec. The simple built-in pre-processors mostly take a text line as input and tokenizes it keeping only the words between given minimum and maximum length. Grouping tags into sentences separated by newline is a practical way of converting tagging data into a pre-processor's friendly format. In this case, there is no punctuation to be removed since ARTigo's system does some validation already while collecting the tags.

Collobert et al. propose replacing all numbers found in the training corpus with the string *NUMBER* [4]. Additionally, they replaced all words that are not in a vocabulary containing 100.000 words by the string *RARE*. These both procedures seem to help filtering out out-of-the-context words that might disturb the training process. Since ARTigo's vocabulary size is only near the half of the one in [4], this last replacement was not executed. In this work, all the tags composed exclusively by digits in ARTigo are ignored. This decision has been made once that, semantically seen, is hard or impossible to measure the similarity between a number and a word. Besides that, ARTigo users will rarely search for numbers. If so, a match can be tried with the *year* field of each artwork.

In the proposed adaptation of doc2vec, a ”document” represents an artwork (id, tags). It additionally requires the document IDs (artwork IDs) for its training, so these are extracted into a separate file during the pre-processing of the tags. Considering a minimum corpus size required for meaningful training (as mentioned in chapter 2.5), this project will cover only ARTigo taggings in German.

3.2 Word2vec

Word2vec is a shallow, two-layered neural network that produces word embeddings, in which words or phrases from a text corpus are mapped to vectors of real numbers that represent the relationships between them. This neural network is divided into two model architectures for learning distributed representations of words that try to minimize computational complexity [10]. In both models (CBOW and skip-gram), a context window of a given size is moved through the corpus. In every step, the model is trained on the words within this context window (see Figure 3.1).

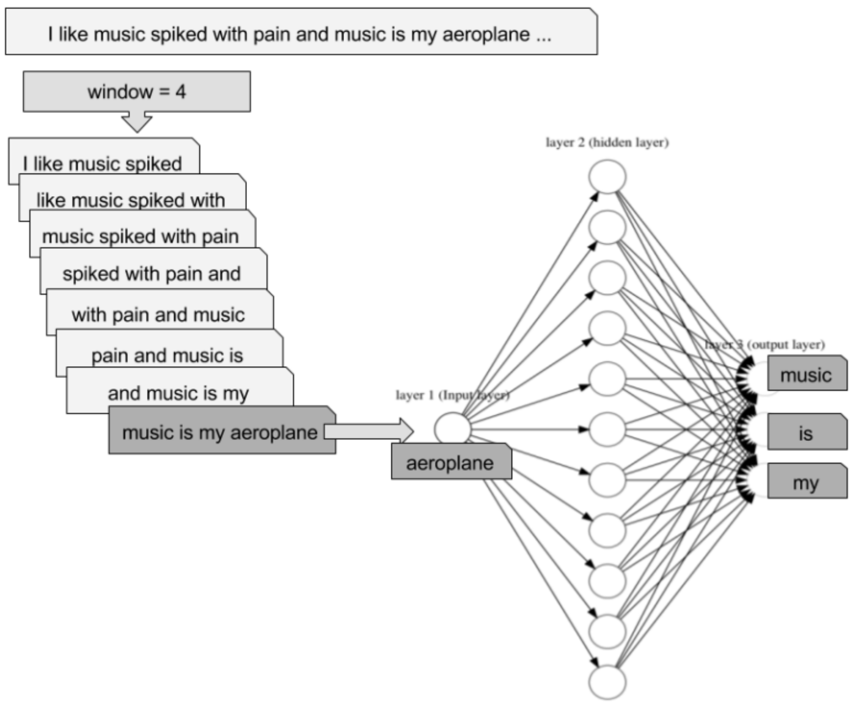


Figure 3.1: word2vec’s distributional representation of words in a neural network
Image source: `https://jaxenter.com/deep-learning-search-word2vec-147782.html`

3.2.1 Parametrization

The word2vec model can be influenced by changing its training parameters, the most relevant are:

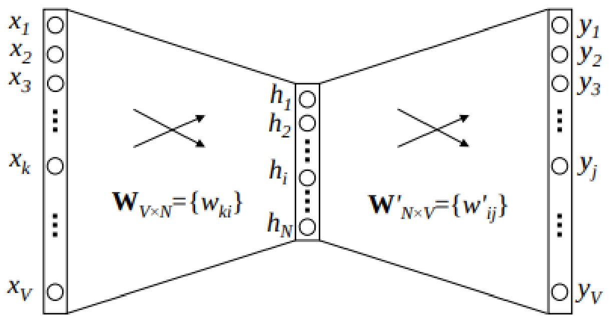

x_1
x_2
x_3
x_k
x_V
h_1
h_2
h_i
h_N
y_1
y_2
y_3
y_j
y_V
$\mathbf{W}_{V\times N}=\{w_{ki}\}$
$\mathbf{W}'_{N\times V}=\{w'_{ij}\}$

Figure 3.2: word2vec's CBOW model
Image source:
`https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/`

`sg` training algorithm: 1 for skip-gram, otherwise CBOW;

`hs` if 1, hierarchical softmax will be used for model training. If 0, and negative is non-zero, negative sampling will be used.

`min_count` ignores all words with total frequency lower than this;

`alpha` the initial learning rate;

`window` maximum distance between the current and predicted word within a sentence;

`negative` if > 0, negative sampling will be used, the int for negative specifies how many "noise words" should be drawn (usually from 5 to 20). Noise words contain for example typographical errors or colloquialisms and thus lowering the quality of the data being processed. If set to 0, no negative sampling is used;

`vector_size` dimensionality of the word vectors.

In chapter 4.1, the investigated parameters are discussed based on experiments varying their values.

3.2.2 Training word vectors

In this paper, only CBOW model is applied, seeing that it is faster and delivers better results for bigger datasets [10] and folksonomies [14] comparing to skip-gram and other word embedding algorithms. As unlike standard bag-of-words model, it uses continuous distributed representation of the context [10] (see Figure 3.1). CBOW trains on sequences of words to predict a word from its context, i.e., from its neighboring words in a given context window.

As illustrated on Figure 3.2, the input layer consists of the one-hot encoded input context words x_1, x_C for a word window of size C and vocabulary of size V. The hidden layer is a N-dimensional vector h. Finally, the output layer corresponds to the output word y which is also one-hot encoded. The one-hot encoded input vectors are connected to the hidden layer via a $V \times N$ weight matrix W and the hidden layer is connected to the output layer via a $N \times V$ weight matrix W'.

It first needs to be understood how the output is computed from the input (i.e. forward propagation)[1]. The following assumes that the input and output weight matrices (see their

[1]Explanation based on the tutorial: `https://iksinc.online/tag/continuous-bag-of-words-cbow/`

details in [10, 11]) are known. The first step is to evaluate the output of the hidden layer h. This is computed by

$$h = \frac{1}{C} \cdot W \cdot \left(\sum_{i=1}^{C} x_i \right)$$

which is the average of the input vectors weighted by the matrix W. It is worth noting that this output computation of the hidden layer is one of the only differences between the continuous bag-of-words model and the skip-gram model (in terms of them being mirror images). Next, it computes the inputs to each node in the output layer

$$u_j = v'^{T}_{w_j} \cdot h$$

Where $v'w_j$ is the $j_t h$ column of the output matrix W'. And finally it computes the output of the output layer. The output y_i is obtained by passing the input u_j through the soft-max function.

$$y_i = p(w_{y_j} | w_1, ..., w_c) = \frac{exp(u_j)}{\sum_{j'=1}^{V} exp(u_j)}$$

The weight matrices and their computation are explained and referenced in [10] as well as in [11].

Word2vec's goal is to learn to represent each word in a document as a vector of numbers such that words that appear in similar context have vectors that are close to each other. So, in the CBOW model, the intention is to be able to use the context surrounding a particular word to predict the particular word. It does this by taking millions of sentences in a large corpus and for every word it takes the surrounding words (few on the left and few on the right). Then it feds these particular context words (after encoding them as a simple one-hot vector) into the neural network and predicts the word in the center of this particular context. Word2vec trains the neural network and finally the encoded hidden layer output represents the embedding for a particular word. It so happens that when word2vec trains over a large number of sentences, words in similar context get similar vectors.

3.2.3 Building synonym lists

The reason for applying synonym lists generated by word embeddings is that the computational cost of their indexing and retrieval are very cheap. Also, synonym lists are mostly very easy to integrate in every search engine API.

Once the word2vec model is trained, one is able to retrieve the n most similar words for every word in the corpus. In this case, the most predicted words are not necessarily synonyms, but are expected to have strong relatedness with the given word within the corpus. For example, the words *ocean* and *sea* can be considered synonyms, and the word *waves* can be seen as a related word to both of these words. In the current project, while generating Solr's synonym lists, the 500 most frequent words in the corpus have been taken with the 3 most similar words each as synonyms. The focus of this bachelor thesis is on the accuracy of word2vec results and their contribution for learning word embeddings from tagging data. These embeddings are meant to be optimized for a search engine.

In Solr's synonym file, each of the 500 words is specified as two comma-separated lists of related words with the symbols "=>" in between *small => small,tiny,teeny,weeny*. The process of considering these lists of synonyms in the search can be done by Solr's synonym graph filter. If the token matches any word on the left, then the list on the right is substituted. The original token will not be included unless it is also in the list on the right[2], so the

[2]Apache Solr's developer guide: `https://lucene.apache.org/solr/guide/6_6/filter-descriptions.html`

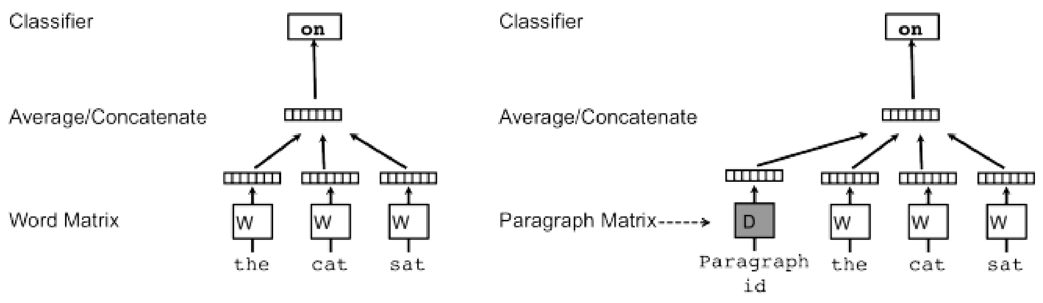


Figure 3.3: word2vec's CBOW model (left) and doc2vec's PV-DM model (right) [8]

original word needs to be added into its own list. There are also other ways of configuring the synonym filter that can bring better search results. This could be experimented in a possible continuation of this work.

3.2.4 Averaging word vectors into document vectors

One of the methods of integrating results from word2vec with a search engine is to average its word vectors into document vectors for representing the artworks (documents). This process is done by taking the vector for each word and its co-occurrence's instances contained by a document, and then summing or averaging them results in a document vector that still might hold rich semantic information about words and documents. After getting a vector for each document, also a generalized way of building a query vector must be conceptualized. This depends heavily on the type of dataset that has been trained and on the kind of search engine one is dealing with. A method of generating query vectors for art2vec's word2vec and doc2vec models is discussed in the section 3.4.

3.3 Doc2vec

While word2vec's task is the generation of word embeddings, doc2vec, as the name also tells itself, is assigned to generate document/paragraph embeddings given a collection of documents. Since word2vec sees a text corpus as one unique entity eventually divided in non-labeled pieces, it is not able to provide a direct representation of these pieces without averaging their word vectors. To overcome this problem, doc2vec simply adds a document id (or any other labels wished) into the data structure of its model (see Figure 3.3). It is also formed by a set of two algorithms that rely on concepts similar to CBOW and skip-gram. They are respectively called distributed memory version of paragraph vector (PV-DM) and distributed bag of words (PV-DBOW).

3.3.1 Parametrization

Doc2vec has at least all the previously mentioned word2vec's hyper-parameters, plus another specific ones. However, it does not necessarily need to store word vectors to produce document vectors. In chapter 4, some of the presented parameters are tested with the goal of achieving the best results. Besides those inherited from word2vec, the most interesting parameters in doc2vec are:

dm defines the training algorithm. If set to 1, distributed memory (PV-DM) is used. otherwise, distributed bag of words (PV-DBOW) is employed;

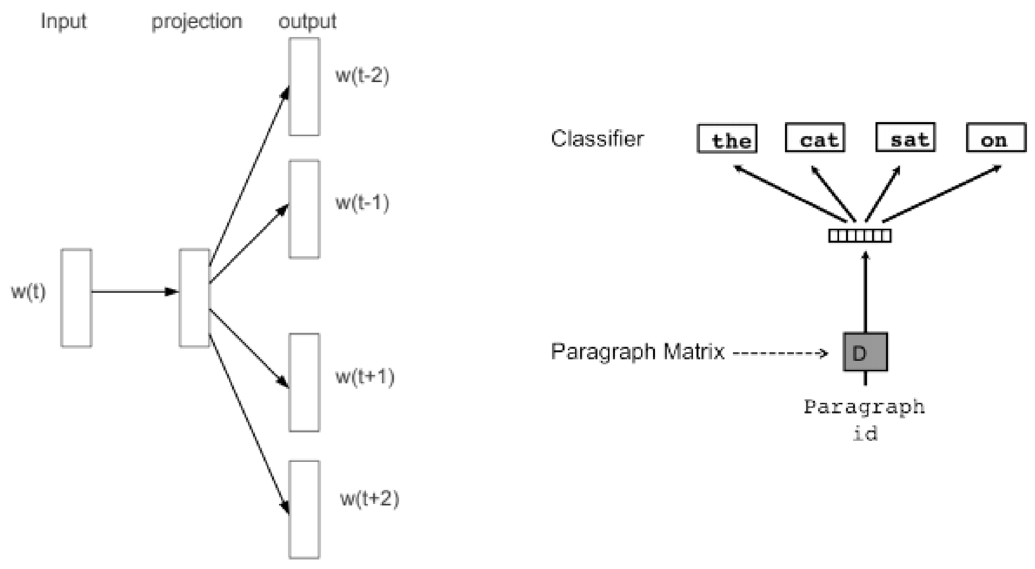


Figure 3.4: word2vec's skip-gram (left) [11] and doc2vec's PV-DBOW model (right) [8]

`dm_concat` if 1, use concatenation of context vectors rather than sum/average; note concatenation results in a much-larger model, as the input is no longer the size of one (sampled or arithmetically combined) word vector, but the size of the label(s) and all words in the context strung together;

`min_count` ignores all words with total frequency lower than this;

`dm_mean` if 0 , use the sum of the context word vectors. If 1, use the mean. Only applies when dm is used in non-concatenative mode;

`max_vocab_size` limits the RAM during vocabulary building; if there are more unique words than this, then prune the infrequent ones. Every 10 million word types need about 1GB of RAM. Set to None for no limit.

3.3.2 Training document vectors

Tagged data does not hold any information about word ordering, thus they cannot easily represent meanings between each other within a document. Alike in the CBOW model, in PV-DBOW the word order within a document also does not have any influence on the projection layer. Taking into consideration the different approaches of both models' architectures while training the data, the more obvious decision is to chose PV-DBOW for this project. It has also the advantage of using less memory than PV-DM, since it ignores the context words in the input, but force the model to predict words randomly sampled from the document in the output. In another perspective, the model architecture of PV-DBOW can be compared with skip-gram's (word2vec), seeing that both use distributed representation of words (see 3.4).

PV-DBOW only needs to store the softmax weights as opposed to both softmax weights and word vectors in PV-DM. It is important to notice that PV-DM delivers state-of-the-art results for most tasks, due to the use of the computed word vectors which contain the semantics of the words. Also, PV-DM considers the word order, at least in a small context, in the same way that a n-gram model with a large n would do.

3.4 Generating query vectors from search requests

When a user type a search query, the query terms need to be tokenized and passed into a pre-trained model of word2vec or doc2vec. In doc2vec's case, the model predicts the vector representation for a query containing these words based on the existing document vectors. This functionality of doc2vec has actually the purpose of inferring a vector given a post-bulk training document. Therefore, the inferred vector representation is in the document space and not in the query space. Hence, using this vector to compare with documents by means of cosine similarity probably does not deliver the results expected in a search engine. The reason for that is the word (tag) frequency in an ARTigo document (image) and the number of unique words in a document which is much greater than the number of words in a search query. Seen that, a comparison between the inferred document vector with the rest of the trained vectors is likely to result in documents with similar size of the query's. In word2vec's case, a document vector cannot be directly inferred, so the query vector must rely on pure averaging techniques such as sum and mean. Through this process, a query vector built upon similarities from word2vec would also not be semantically rich enough to compare with the document vectors.

Algorithm 1: proposed general query vector generation for doc2vec models

```
T := (term_1, term_2, ..., term_n) ;                          // input, set of terms
Q := null ;                                                    // output, query vector
M := null ;                                // set of N similar terms for each term in T
E := null ;                  // set of expanded query terms with their similar terms M
C := null ;                              // set of final query terms with their repetitions
N := 3 ;                                     // number of most similar terms to be taken
foreach $t \in T$ do
    most_similar := word2vec.most_similar($t$, N) ;          // take the N most similar
    M := M $\cup$ most_similar ;                                 // terms from word2vec
end
E := T $\cup$ M ;                                                // build the expanded term set
foreach $t \in E$ do
    freq := average_tfidf($t$) ;                                    // ((TF-sum) / IDF)
    while $freq > 0$ do
        C := C $\cup$ $t$ ;                        // insert copies of each term into the set
        freq := freq - 1 ;           // based on its average co-occurrence frequency
    end
end
C := shuffle(C) ;                                   // randomize the order of the elements
Q := doc2vec.infer_vector(C) ;          // pass the expanded query terms into doc2vec
return Q ;                                  // return the inferred document/query vector
```

In order to bypass this undesirable effect of inferring poor query vectors from word2vec and doc2vec, a couple of empirical thoughts can be made. If a user inputs a longer query with similar terms and thus expanding its context, the results in this approach will probably get more accurate. To experiment that, in a typical ARTigo search query which is often short, for every query term, one can retrieve the most *n* similar words from a pre-trained word2vec model on ARTigo's dataset and then append them to the set of query terms. This bigger set of words that represents a longer document has to be given as input into doc2vec's inferring method, or in word2vec's case, the average of their vectors is computed. Comparing the predicted vector for this tuned search query with the document collection will very likely match results with a greater similarity to the initial query than using only the original search input to infer the query vector. However, this term expansion has to be adapted for search queries that contain more than three terms. Otherwise, too many related words for each term would expand the context of the query too widely, and thus loosing its original meaning.

Another crucial difference between what a user enters in a search query and what it is in the documents is the frequency in which a word occurs in both. Search engine users are not intended to enter repeated words in their queries. On ARTigo, many different users tag the same image with the same word. Consequently, a big amount of documents are composed by words with high co-occurrence frequency within an individual document. That being the case, the set of query terms can be expended with repetitions of them. For each particular query term, the number of repetitions can be taken from its co-occurrence average by document in the whole dataset. Such average can be calculated by taking the sum of the term frequency (TF) for each document in the whole collection and dividing it by the number of documents in which the particular term occurs (IDF). These both values can be easily requested to Solr and used for this purpose. Another constraint is the number of repetitions of the original query terms. It has to be balanced with the highest frequency in the predicted set of terms. Otherwise the original terms might be shadowed by high frequent similar terms. Therefore, the number of repetitions for an original term has to be equal or greater than the highest frequency in its set of predicted similar terms.

After going through these described steps (see Algorithm 1), it is clear that in the end a query is being translated into a typical document from the dataset. For example, a query *sea ship* could be expanded to a query similar as *sea sea sea ocean ocean ocean waves waves ship ship ship ship boat boat marine*. Since the query is now an ARTigo-like document, comparing it with the collection makes much more sense. The same process can be used for generating query vectors to score word2vec models. Experiments on this topic are detailed in chapter 4.

3.5 Ranking documents using vectors

After getting the results from word2vec and doc2vec, these could be integrated into ARTigo's search engine. In word2vec's synonym lists case, Solr's native synonym graph filter takes care of the integration. For doc2vec's and word2vec's vector approaches, a custom scoring algorithm is required for retrieving the documents based on the similarity of their vectors with the query vector generated in the section 3.4.

A recently developed and by now not wide used Solr plugin[3] covers vector scoring in a minimum required way for testing the results obtained from the trained models in this thesis. With this plugin, documents can be ranked using dot product or cosine similarity given a query vector of same dimension as a document vector. However, scoring a relative big collection of documents (> 68K) trained with an efficient vector size (approx. 500) can take a very long time comparing with TF-IDF (even with synonym lists) scoring on a term-document matrix. Therefore, this thesis aims to experiment with different corpus and vector sizes in order to obtain acceptable results with the best computational cost (see chapter 4).

[3]Solr vector scoring plugin: `https://github.com/saaay71/solr-vector-scoring`

CHAPTER 4

Experiments and evaluation

Recently, experimenting with word and document embeddings has become a very easy and practical task to perform thanks to Mikolov's work. The world has already experienced almost *anything2vec*[1], meaning that word embeddings have been applied on data of almost any topic (medical diagnosis, psychology, biology, etc). However, there is still place for research with word2vec on tagging data, specially in the context of artworks. This project mainly analyzes the parametrization of word2vec and doc2vec for the source corpus. Starting this analysis, the parameters of the algorithms are varied in an acceptable range towards the best performance. In each experiment, the generated models are automatically evaluated after the training in an unsupervised manner. A good and supervised evaluation alternative for word2vec models could be as described in [19]. Finally, the applied word2vec and doc2vec embeddings on ARTigo's dataset are evaluated by different groups of users, in a supervised way inspired by Schneider [20].

4.1 Word2vec's embeddings

4.1.1 Evaluation method

For all experiments, the evaluation is based on the same set of syntactic and semantic questions for the German language that have been created by Müller [13][2]. These questions are actually almost translations of the same questions in the English language offered in word2vec's framework. In the aspect of word embedding evaluation, semantic and syntactic questions have the following form: does *man* relate to *woman* as *vater* relates to *mother*? If the difference between the similarity scores of *man-woman* and *vater-mother* is minimal, then the answer to the question is *yes*. The more *yes* answers a model gets, the more accurate it is. Basically, a *yes* hit is produced if the similarity pattern between two word pairs with same kind of relation is near of being identical. Such similarity patterns on ARTigo's tags can be observed in Figure 4.1. For example, it is noticeable that female words are in the positive value range and the male are in the negative range (y-axis). Other interesting observations can be made among country and language names. In the vector space, languages are situated near to their respective countries. Also, one can observe that Asian

[1]NLP Town's blog on word embeddings: `http://nlp.town/blog/anything2vec/`
[2]Work produced by the cited bachelor thesis: `https://devmount.github.io/GermanWordEmbeddings/`

countries (or languages) from ARTigo's data such as *china* and *japan* are located in the positive range while European countries like *deutschland* and *italien* are in the negative (x-axis). One can also notice that, on ARTigo's vocabulary, the typo *mnnlich* is surprisingly nearer to the pair *weiblich* than its correct spelling (*männlich*).

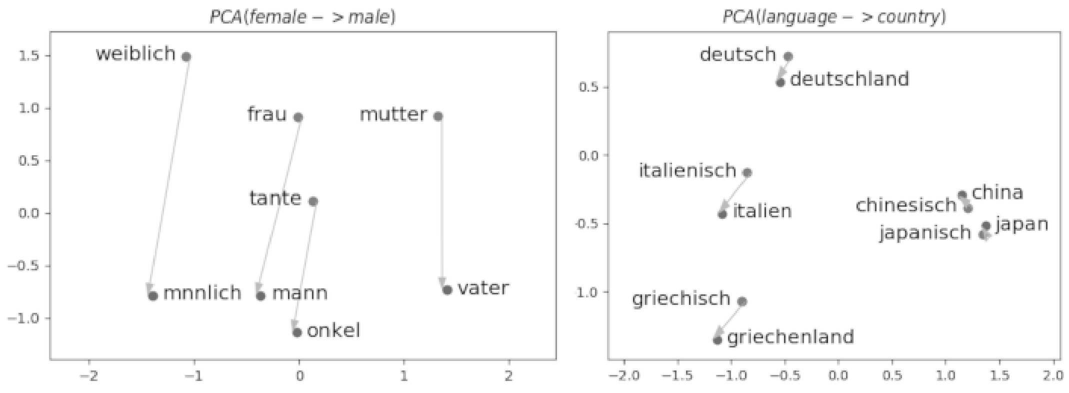


Figure 4.1: position of semantic word pairs in word2vec's vector space (ARTigo dataset) - plotted using PCA (principal component analysis)

The work elaborated in [13] offers syntactic questions for nouns, adjectives and verbs. The main purpose of syntactic questions is to check the relation between different variations of the same word, for example *woman* and *women*. In German, nouns and adjectives vary in *genus*, *numerus* and *casus*. Adjectives can also be raised into *comparative* and *superlative*. Detecting these kind of linguistic features is very meaningful for general text data such as Wikipedia. That way, it is possible to find out if a verb in its past tense is used in the same contexts of its present or future tense. As ARTigo's dataset is composed mainly of nouns, the verbs and the adjectives have been removed from the syntactic evaluation process. Indeed, it would otherwise deliver only bad results in these categories for all the trained models. The selected syntactic questions are composed by **singular => plural** relatedness of nouns, totalizing 1000 samples.

The semantic questions are grouped in *opposite* (300 samples), *best match* (540) and *does not fit* (110). While *best match* questions express relations of the kind **country => capital** or **masculine => feminine**, *does not fit* questions attempt to check relations between pairs of words belonging to the same category, such as **windows => linux** or **city => city**. Semantic *opposite* questions have the form **positive => negative**. To analyze the generated models by every parameterization, an existing word2vec evaluation tool for the German language [13] was adapted.

Although one group of syntactic questions is evaluated, it makes more sense to guide the evaluation based on semantic questions such as *best match* and *does not fit*. The motivation for this lays on the lack of ARTigo's dataset capability of representing noun relatedness like the suggested singular-plural. The simple cause of this problem is that there are not enough nouns in the dataset with their morphological/syntactic variations of *numerus*. Almost the same happens with semantic *opposite* words. Only semantic features that do not require the pair to be a specific word are able to be successfully evaluated. Hence, semantic *best match* and *does not fit* questions are very good for evaluating this particular type of dataset. Nevertheless, ARTigo's vocabulary (*min_count*=150) covers no more than 20% of the testset created for Wikipedia [13]. With that in mind, it is important to build a specific testset to improve the quality of the evaluation.

4.1.2 Pre-processing

Not ignoring the fact that the distribution method of the tags within an artwork does affect the values of the predicted word vectors, this work has developed a pre-processing tool whose parameters are described in section 3.1. Such an experiment was left in [14] for future work, so this evaluation is now required.

accuracy per pre-processing configuration (ARTigo dataset)				
configuration	best match	does not fit	top 10 nouns	opposite
O-O-O-O	5.3%	48.5%	20.2%	2.5%
O-O-S-O	11.1%	48.5%	8.7%	**6.3%**
O-O-S-S	8.0%	48.5%	12.5%	1.7%
C-O-O-O	10.6%	42.2%	**45.4%**	2.7%
C-O-S-O	7.8%	53.1%	4.8%	5.8%
C-O-S-S	18.4%	67.2%	16.9%	4.3%
C-20-S-S	**20.6%**	**70.3%**	15.1%	1.6%
C-30-S-S	16.3%	**70.3%**	17.9%	5.0%
C-35-S-S	18.1%	62.5%	16.7%	4.7%

Table 4.1: word2vec's model accuracy for each pre-processing configuration: co-occurrences (C: with; O: only unique tags) - maximum (*x* (e.g. 30): limit the co-occurrence frequency based tag count to a maximum of *x*; O: co-occurrence until tag count) - shuffle 2x (S: shuffle; O: do not shuffle) / trained on CBOW with default parameter values

The performed configurations are: one model without co-occurrence (only unique words for each document), and other with co-occurrence based on tag count. For each of the co-occurrence configurations, one sample without shuffle, one with *outer* and another with both *outer* and *inner* shuffling is created. If the model trained that outperforms the other 5 models is a co-occurrence-based one, then the maximum co-occurrence parameter is tested with the values of (20, 30, 35). This leads to a total of 9 differently organized document structures. All these word2vec models for ensuring pre-processing quality have been trained with the default values: *min_count*, *window* and *negative*=5, *alpha*=0.025 and *vector_size*=100.

In general, the models without co-occurrence have been the worst to perform (see Table 4.1). Surprisingly, the model with co-occurrence but without any shuffling has outperformed all the others in the syntactic questions. Since the present evaluation does not rely on this category, the next models have been trained. Up to this point, the model with co-occurrence and with *outer* and *inner* shuffle has performed the best results. Then, parting from this configuration, variations for the maximum co-occurrence parameter have been executed. It turns out that a maximum co-occurrence of 30 is the ideal for this data, so as this number appears to be in a document the minimum frequency of at least 1.7% of the vocabulary (521 out of 30.058 unique words, *min_count*=5).

The parameter configuration in the pre-processing that achieves the best results is composed by word co-occurrence based on tag count with maximum co-occurrence of 30 that apply both *outer* and *inner* word position randomization. This proves that if the occurrence position of a word in its belonging sentence is random, then word2vec could deliver better accuracy than with words in alphabetical ascending order. Due to the fact that, in this case, co-occurrence datasets have outperformed all the others, the statement that word2vec fails to take advantage of the repetitions present in the data [17] can sound a bit ironic. That observation was however made regarding word2vec's context window against word frequency in other models such as LSA.

4.1.3 Minimum count

Now that the right pre-processing is discovered, it is time to experiment with word2vec's parameters. The first one to be examined is *min_count*, which restricts the use of words with frequency equal or higher than a given threshold. Interesting is that with *min_count* set to 2, the corpus size reduces to impressing 52% of the original, retaining 52.222 unique words and dropping 47.393. These numbers prove that almost the half of the taggings appear only once in the entire dataset. Such phenomenon could be explained by analyzing the habits of ARTigo's users. Seeing that the users have mostly 60 seconds to play one round and are often playing against another user, they are invited to challenge themselves by entering as many words as they can think of (if they have competition spirit, of course). Thus, they are under time pressure and possibly do not put much attention on the quality of what is being entered. Another feature within the GWAPs is the user's score, which motivates the user to enter more expressive words. A very improbable reason for half of ARTigo's tags being unique is that many of the players do not care about the score they are achieving. A much more imaginable cause for the word uniqueness frequency could simply be the immense diversity of the vocabulary that composes the many different topics included in the vast context of art. Isolated context-specific images that have not been tagged with a considerable frequency could also contribute for such corpus characteristic.

In this work, nine models are trained with *min_count* varying in (5, 15, 30, 50, 75, 100, 150, 175, 200). From 1 to 200, the number of tokens drops mere 11% (leaves 7.989.891 from 8.891.052), while the vocabulary size reduces shocking 97% (retains 3.573 out of 99.615 unique words). These numbers mean that the most frequent tags in ARTigo compound the majority of the tag assignments. This shall be good for arising a very rich semantic relatedness between these frequent words. In the experiments, a minimum count of 150 has presented the best accuracy with up to 85.7% at *does not fit* and 49.9% at *best match* semantic questions. Around 4% (4.271) of the vocabulary size with 91% (8.109.754) of the corpus size would remain for a final training corpus. Considering that the corpus in the fact is big enough, this amount of words would be applicable for a purely semantic task such as predicting synonyms. However, the corpus is being used to implement a search engine based also on document vectors.

To efficiently retrieve documents, a search engine needs more than 4.271 keywords in its index if the corpus size is not small. The word/document vectors work in this case as a reduced index. If the term that is being searched was not previously trained by word2vec, then the search engine cannot retrieve documents containing this term. This happens because the searched term, that could have been in some documents, appears in the entire corpus less frequent than the number of times given in *min_count*. Hence, the *min_count* parameter needs to be refined towards the best precision and recall of the search engine's results. Since the evaluation in this thesis does not cover the precision of the search engine's results, a *min_count* parameter of 150 has been kept for all word2vec and doc2vec models.

4.1.4 Learning rate and context window

The next tests are about training 10 models with different learning rates (0.01 += 0.01 until 0.1). In contrast with [14], the best initial learning rate (*alpha*) for ARTigo's dataset seems to be 0.03 (Table 4.2 (b)), close to the default of 0.025. Another intriguing affirmation is that it could be a problem if the window size for a non-linear corpus is chosen too small [14]. In their work, the perfect value pops up being the default of 5. To prove that, 7 different models have been trained with the window sizes of (3, 5, 10, 15, 20, 25, 30). In the executed experiments, it is interesting to notice that the longer the context window is, the better the accuracy for all tasks but *opposite* semantic questions (see Table 4.2 (c)). The context window size that has outperformed is 25, and therefore this value has been chosen for the final training.

accuracy per parameter configuration (ARTigo dataset)				
parameter	best match	does not fit	top 10 nouns	opposite
(a) *min_count*				
5	19.9%	60.9%	15.8%	3.9%
15	24.8%	72.5%	25.5%	5.9%
30	24.9%	66.0%	30.8%	7.1%
50	37.7%	69.4%	37.9%	7.0%
75	40.2%	75.9%	38.6%	7.9%
100	**50.0%**	75.0%	39.3%	9.4%
150	49.4%	**85.7%**	49.4%	**11.7%**
175	47.2%	81.0%	**53.4%**	10.5%
200	46.1%	73.7%	48.4%	10.5%
(b) *alpha*				
0.01	21.6%	60.9%	16.0%	4.3%
0.02	19.9%	62.5%	16.9%	3.9%
0.03	**21.6%**	59.4%	15.8%	**5.4%**
0.04	20.6%	62.5%	17.6%	4.3%
0.05	18.4%	60.9%	16.5%	2.7%
0.06	19.1%	**65.6%**	17.4%	3.5%
0.07	20.2%	**65.6%**	17.4%	4.3%
0.08	18.8%	64.1%	16.2%	3.5%
0.09	19.1%	60.9%	**17.9%**	4.7%
0.10	17.0%	62.5%	16.7%	2.7%
(c) *window*				
3	12.8%	59.4%	17.2%	3.1%
5	17.4%	65.6%	18.2%	3.1%
10	17.0%	68.8%	18.2%	5.8%
15	17.0%	**76.6%**	20.5%	**7.0%**
20	19.1%	71.9%	20.8%	4.7%
25	20.2%	75.0%	**21.3%**	5.4%
30	**20.9%**	**76.6%**	19.1%	3.9%
(d) *negative*				
2	15.6%	71.9%	16.0%	1.2%
3	16.7%	**73.4%**	17.4%	2.7%
5	**19.9%**	60.9%	15.8%	3.9%
10	18.1%	68.8%	18.6%	3.9%
15	18.8%	68.8%	19.3%	5.0%
20	17.4%	67.2%	19.4%	4.3%
25	17.4%	68.8%	**20.0%**	**5.4%**
(e) *vector_size*				
50	15.6%	67.2%	15.0%	3.9%
100	14.9%	68.8%	**18.2%**	**4.3%**
150	**17.4%**	**71.9%**	16.2%	3.1%
200	14.2%	68.8%	17.6%	3.9%
300	13.1%	65.6%	16.7%	2.7%
500	11.7%	65.6%	16.0%	1.6%

Table 4.2: word2vec's model accuracy for each parameter, trained on CBOW with default parameter values

4.1.5 Negative samples

For using with its both algorithms, word2vec comes with two optimization strategies: hierarchical softmax and negative sampling. The first one represents all words in the vocabulary in a binary tree and tends to deliver better embeddings for infrequent words. The second option trains based on sampling places it might have expected a word, but did not find one, which is faster than training an entire corpus every iteration. Due to the fact that a *min_count* of 150 (without infrequent words) has been chosen for the final training, this thesis has opted to apply only the negative sampling optimization technique.

Negative sampling (NEG) is a simplification of noise contrastive estimation (NCE), a technique which hypothesizes that a good model should be able to differentiate data from noise by means of logistic regression. The main difference between NCE and NEG is that NCE needs both samples and the numerical probabilities of the noise distribution, while NEG uses just samples [11]. The experiments in [11] indicate that values for NEG in the range 5-20 are useful for training on small datasets, while for large datasets as small as 2-5 can be used. Negative samples have been tested for small and big folksonomy datasets in [14]. Also, in [6], a general and much larger corpus than ARTigo's is trained but with a vocabulary size just twice as big as ARTigo's. In these two papers, the average number of negative samples that produces the best accuracy lies around 20. Actually, the influences of context to negative samples could be negative [6].

Since ARTigo's corpus size can be considered as medium, experiments with the values (2, 3, 5, 10, 15, 20) for negative samples have been executed. In the tests listed on Table 4.2 (d), the experimented numbers of negative samples have all delivered similar results. However, results for nearby values (3-5-10) seem to differ in more than 10% on semantic tasks such as *does not fit*. Therefore, the best amount of negative samples is 15, with which the accuracy has stabilized for every task.

4.1.6 Vector size

The last parameter experiments are about the number of dimensions of the word embedding vectors. To ensure the reliability of the results, six models have been trained with vector sizes in the span (50, 100, 150, 200, 300 500). As shown in Table 4.2 (e), the best vector dimension of 150 is comparable with the one of 100 (default) in [14]. This confirms that not for every corpus size or context a higher vector dimension performs the best results. One could also assimilate these results to how word2vec behaves for each corpus size regarding the given *vector_size*. Comparing Wikipedia's corpus size with ARTigo's, and then their respective outperformed vector dimensions with each other, it makes sense that the appropriate *vector_size* for ARTigo's dataset is 150 and not some value near to 500.

4.2 Doc2vec's embeddings

4.2.1 Evaluation method

Unlinke word2vec's, doc2vec's results are much more difficult to evaluate. That depends on the type of dataset and on the task being implemented. Doc2vec takes labeled documents as input. That means it can be used for many classification tasks. Since the problem pointed by this thesis is not about classifying documents, the only label used in each document is its ID. Here, the goal is simply to generate a vector for each document so that one can compare the similarities between them. So, there is one and only practical way of evaluating how good these vectors represent the documents. Sample documents of different lengths from several contexts, retrieve their most, median and least similar and then compare how much the vocabulary is been shared.

shared vocabulary per algorithm and averaging (ARTigo dataset)			
parameters	most	median	least
(a) abstract			
dm=0	7.89%	7.59%	8.31%
dm=**1**, *dm_mean*=**0**	6.37%	7.99%	**0.60%**
dm=1, *dm_mean*=1	8.13%	7.38%	8.65%
(b) mixed			
dm=0	10.09%	7.99%	8.54%
dm=**1**, *dm_mean*=**0**	7.76%	8.45%	**0.59%**
dm=1, *dm_mean*=1	9.85%	8.30%	9.27%
(c) concrete			
dm=0	8.06%	7.33%	7.80%
dm=**1**, *dm_mean*=**0**	6.35%	7.47%	**0.36%**
dm=1, *dm_mean*=1	8.21%	7.31%	9.07%

Table 4.3: percentage of shared vocabulary between 100 samples (of 3 different contexts) and their 10 most-median-least similar documents in each of the parameter configurations for doc2vec

For the optimization of doc2vec's hyper-parameters that have been tested, a set of documents have been selected. These documents are grouped into three principal context categories: abstract, mixed, concrete. In each of this categories, a sample of 100 random documents have been taken. For analyzing each sample, the 10 most, median and least similar documents have been retrieved from doc2vec's similarity scores.

4.2.2 Optimized hyper-parameters

Doc2vec has almost all hyper-parameters offered by word2vec. Testing and evaluating all these parameters again with doc2vec would exceed the time available for this project. Only for that reason, it was decided to inherit the optimized parameter values from the experiments in section 4.1. Note that re-running these parameter tests for doc2vec would probably boost the recall of the final results in the search engine.

Training algorithm and word vector averaging

The specific hyper-parameters of doc2vec that have been tested concern only the training algorithms. PV-DBOW has no optimization options while PV-DM has two of which only one has been tested. Similarly to word2vec's averaging approach, this tested parameter (*dm_mean*) defines if the word vectors should be summed or averaged. As shown in the Table 4.3, only the PV-DM algorithm with summed word vectors has been capable of differing between most and least similar documents on ARTigo dataset.

Vector size experiments

Both word2vec and doc2vec have the *vector_size* parameter. It defines how many dimensions a vector that represents a document should have. The best hyper-parameter values from last section have been taken to optimize the *vector_size* parameter. Looking at Table 4.4, one can make many observations. Most similar documents are well recognized by low vector dimensions, while least similar documents are effectively dropped out through high vector dimensions. All vector dimensions have been able to capture median similar documents. Since the goal in this thesis is to find the lowest vector dimension with the highest

shared vocabulary per *vector_size* on PV-DM with sum averaging (ARTigo dataset)			
vector_size	most	median	least
(a) abstract			
50	**8.69%**	**7.60%**	4.90%
100	8.07%	7.45%	2.77%
150	6.58%	7.58%	0.84%
200	5.65%	6.93%	0.07%
300	3.95%	7.74%	**0.00%**
500	2.21%	7.49%	**0.00%**
(b) mixed			
50	**11.28%**	8.06%	4.56%
100	10.28%	8.22%	2.42%
150	8.62%	7.88%	0.72%
200	7.37%	8.35%	0.12%
300	5.39%	8.32%	**0.00%**
500	3.62%	**8.98%**	**0.00%**
(c) concrete			
50	**9.16%**	7.39%	4.83%
100	8.42%	7.27%	2.70%
150	7.71%	7.80%	0.83%
200	6.21%	7.73%	0.09%
300	3.92%	7.93%	**0.00%**
500	2.21%	**7.96%**	**0.00%**

Table 4.4: percentage of shared vocabulary between 100 samples (of 3 different contexts) and their 10 most-median-least similar documents in each of the values for doc2vec's *vector_size* parameter

accuracy, the *vector_size* of 150 has been chosen for the recall evaluation.

4.3 Art2vec's search engine evaluation

The last evaluation step in this bachelor thesis is about comparing the results of the different word embedding approaches with themselves. Since ARTigo's current search engine already reaches a good precision (true positives), this project only covers evaluation of the recall. To understand recall, one can make this simple question: *how many relevant documents have been retrieved?* The less false negatives are miss-recognized, the better the search engine's recall. Another meaningful reason for evaluating the recall is that the applied word embedding techniques generate a similarity vector space for the vocabulary, which has much to do with retrieving documents that do not contain any of the query terms. Quite exactly these documents are not retrieved by ARTigo's current search engine.

In this work, a search engine for tagged artworks has been built based on three different approaches on word embeddings. The first method is to use word2vec's trained model to build synonym lists which are integrated into Solr's TF-IDF scoring. The second averages this trained word2vec model to build document vectors and then use them to compare with a query vector. While generating a query vector for word2vec, each query term is expanded in more 3 similar terms (according to word2vec's predictions). This step is also applied by the proposed query vector generation (Algorithm 1) for doc2vec models. This algorithm consists of the intertwining between doc2vec's generated document vectors, word2vec's most similar words and the average occurrence of a term per document (TF-IDF variation).

accuracy by approach and tag category (art2vec built on ARTigo dataset)			
category	synonyms	word2vec	doc2vec
combination	31.48%	**49.07%**	14.20%
specialization	29.32%	**44.75%**	15.74%
deep semantics	58.02%	**77.16%**	21.60%
surface	67.59%	**67.90%**	26.85%
AVERAGE	46.60%	**49.46%**	19.60%

Table 4.5: percentage of relevancy of the top 12 artworks (potentially recognized as false negatives by ARTigo) that have been retrieved by art2vec (recall evaluation)

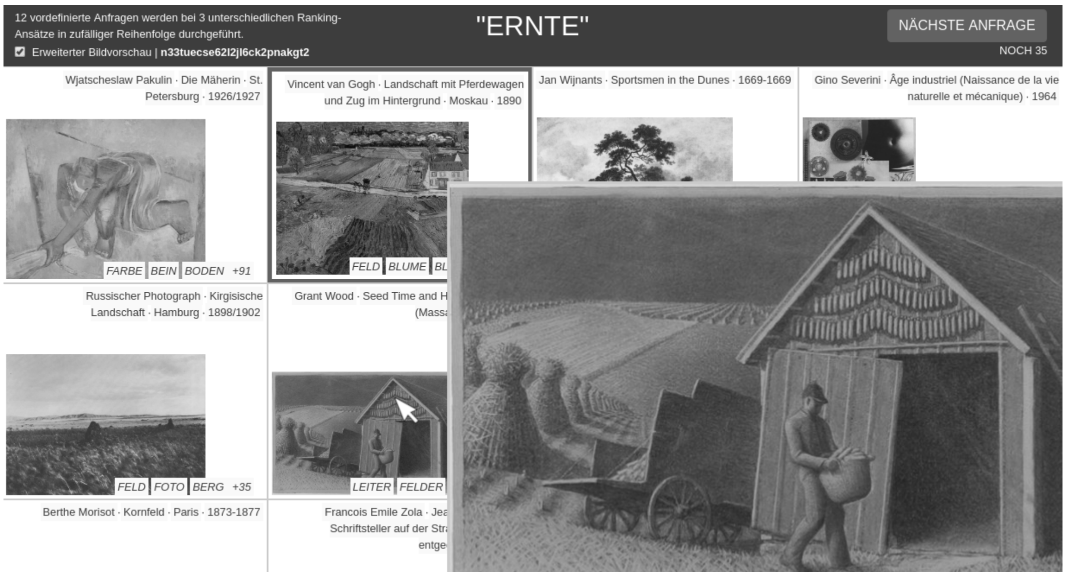
To generate query vectors for word2vec models, the average of all word vectors from the expanded query is taken.

For evaluating the recall of word2vec's approaches, the following parameters have been used for the training: *sg*=0, *hs*=0, *min_count*=150, *alpha*=0.030, *window*=25, *negative*=15, *vector_size*=150. It is important to notice that a word2vec model for the implementation of a entire search engine should be trained with a *min_count* much less than 150, as the search can then reach a bigger vocabulary. The best parametrization experimented for doc2vec (PV-DM with summed word vectors) have been used for the evaluation of its final trained model in the search engine. All the other doc2vec parameters inherited from word2vec have been set to the best values achieved on the previous word2vec experiments.

Before the recall evaluation, a set of 4 categories of search queries have been defined to divide them in: combination (e.g. *ritter schlacht, volk strasse*), specialization (e.g. *stadt dunkel, wetter heiss*), deep semantics (e.g. *traurig, durcheinander*) and surface tags (e.g. *mensch, landschaft*). Each category contains a total of 3 different queries. All these predefined queries have been executed for each of the implemented techniques. For this evaluation, a specific user interface has been designed (see Figure 4.2). A total of 9 different users have participated and selected the documents they have judged as relevant for each of the executed queries. They could whether select all documents in the result set (max. of 12 results per query), a few, or none of them. The order in which each query (with its particular ranking technique) shows up has been completely random.

As shown in Table 4.5, word2vec's vector scoring has been the approach with the highest overall accuracy (49.46%) for improving the recall of art2vec. The synonym approach comes right behind with 46.60% accuracy. Doc2vec's approach has been the worst to perform, with an accuracy of only 19.60%. The reason for doc2vec's poor performance might lie on the assumption taken in which word2vec's refined parameters should also fit to a doc2vec model. In this work, the inherited parameters from word2vec have not been experimented along doc2vec, since every model takes around 4 hours to be trained on ARTigo dataset. Nevertheless, the final doc2vec model has shown in short experiments (not reported here) to be a good alternative for the commonly used Rocchio algorithm on the task of relevance feedback.

Analyzing art2vec's evaluation results, one can also see that word2vec has outperformed in all of the 4 proposed tag/keyword categories. Furthermore, the category of deep semantic tags has shown the highest accuracy with 77.16%, which is 19.58% higher than the one achieved by the synonym approach (58.02%). This category represents words with more abstract meaning such as *happy* or *adventure*. Coming next with 67.90%, the surface tags category has reached the second highest accuracy. Words with more concrete meaning such as *person* or *landscape* are related to this category. While in the combination category two words (mostly nouns) have been combined, in the specialization category a noun has been specialized through a second word of type adjective or verb. These both categories have not achieved a minimum accuracy of 50% to offer acceptable quality on


12 vordefinierte Anfragen werden bei 3 unterschiedlichen Ranking-Ansätze in zufälliger Reihenfolge durchgeführt.
Erweiterter Bildvorschau | n33tuecse62l2jl6ck2pnakgt2

"ERNTE"

NÄCHSTE ANFRAGE
NOCH 35

Wjatscheslaw Pakulin · Die Mäherin · St. Petersburg · 1926/1927

Vincent van Gogh · Landschaft mit Pferdewagen und Zug im Hintergrund · Moskau · 1890

Jan Wijnants · Sportsmen in the Dunes · 1669-1669

Gino Severini · Âge industriel (Naissance de la vie naturelle et mécanique) · 1964

Russischer Photograph · Kirgisische Landschaft · Hamburg · 1898/1902

Grant Wood · Seed Time and H
(Massa

Berthe Morisot · Kornfeld · Paris · 1873-1877

Francois Emile Zola · Jea
Schriftsteller auf der Str
entge

Figure 4.2: user interface for art2vec's recall evaluation

document retrieval. Considering the methodology used in this thesis and the presented recall evaluation, once can conclude that a word2vec model trained on ARTigo dataset is actually only applicable for short queries in a search engine. In order to retrieve more accurate results for longer search queries, one needs to try to refine word2vec's parameters in these categories while keeping good results for deep and surface tags. Also, the proposed query vector algorithm needs to be polished.

CHAPTER 5

Outlook

In this bachelor thesis, three different word embedding techniques for building a semantic search engine have been presented. All these techniques rely on generating word and document vectors using word2vec and doc2vec. The approaches differ between synonym lists and vector scoring. Notice that different training parameters are required for these two approaches. The quality of synonym lists depends on the removal of infrequent words in the training set, while a vector-based document retrieval with a wider vocabulary reach relies exactly on these infrequent words.

Although these models can be applied to whatever type of textual dataset, this thesis has focused on folksonomies generated by ARTigo. Since word2vec and doc2vec are both frameworks that were built to work with general structured text data, their parameters have to be refined so that good results can be achieved also for unstructured text such as folksonomies. Hence, the most important hyper-parameters have been tested along a wide range. One of the only assumptions taken in word2vec's parameterization is that its algorithm CBOW works better with folksonomies than skip-gram. Therefore, just CBOW has been applied in the experiments.

Training word2vec with different parameter configurations has led to interesting observations relevant to the fields of IR and NLP. The first experiment looked into the transformation of folksonomy data from an artwork into a single text line. The best attempt composes by repeating a word assigned to an image based on the number of times this image has been tagged with that word (tag count). Also, shuffling the order of all the words within an image has brought the best results. Finally, the maximum co-occurrence of each term within an artwork has been constrained to 30, after evaluating the results with 20, 30, 35 and maximum co-occurrence limited on tag count.

Minimum count is a parameter that behaves differently depending on the goal to be achieved. It works as a threshold for removing words from the vocabulary to be trained that occur with frequency less than the given value. Within ARTigo dataset, in order to generate a stronger semantic relatedness among the most common words in the vocabulary, one needs to increase the minimum count from default of 5 to 150. This can be useful for creating synonym lists and generating query vectors through the proposed Algorithm 1. However, with such a high expressive minimum count, only a small part of the vocabulary is present in the word vectors. Thus, a big amount of the query terms in a search engine would not be found in the document vectors. With that in mind, the default minimum count of 5 could used for both word2vec and doc2vec final document vectors to be stored in a Solr's index.

Another crucial parameter is the vector size. In a general way, it defines how much semantic information each document vector can contain. Nevertheless, it also restricts the viability of the use of these document vectors for implementing a search engine. The higher the vectors' dimension is, the slowest the documents' retrieval. Besides that, a high number of dimensions does not work good for every corpus size and structure. Values between 50 and 500 have been tested and the semantics of each trained model evaluated. It turns out that a vector size of 150 provides the best fit for ARTigo's dataset, which is way smaller than the 500 vector dimensions required for training word2vec on the Wikipedia corpus [12].

The less expressive set of hyper-parameters in ARTigo's case are composed by context window size, learning rate and negative samples. In the executed experiments, the window size that has achieved the best results is 25. Although it has not shown significant difference comparing to the default of 5, this result has confirmed the ones obtained in similar use cases in other publications. Negative samples have stabilized with a number of 15, a bit far from the default of 5. Learning rate has outperformed not far from the default (0.025) with 0.030.

In doc2vec's case, all the inherited parameters from word2vec have been set to the ones that outperformed in word2vec's. Perhaps because doc2vec's PV-DM might store a similar amount of semantic information as word2vec's CBOW, the vector size of 150 outperforms for a doc2vec model trained on ARTigo's tags. However, further research is needed here. In contrast to word2vec, the two offered algorithms have been evaluated. PV-DM has one additional parameter which has been experimented too. The parameter configuration that has outperformed is PV-DM with sum averaging of word vectors.

Using raw document vectors as index in a search engine implies another challenges such as their correspondent vector representation of queries. In most of the cases, queries are much shorter and less expressive than documents. Also, in word2vec's case, query terms have to be replaced by their respective word vectors. These facts make the comparison between queries and documents much more complicated. To overcome the natural lack of semantic similarity between queries and documents, this thesis has proposed an algorithm for translating text queries into word2vec- or doc2vec-like document vectors. The presented method relies on synonym expansion and co-occurrence frequency average (TF-IDF) of the expanded terms. This generation of query vectors deserves deeper and more specific experiments, once it has been implemented with the intent of making the document vectors' evaluation within a search engine possible.

Finally, the evaluation of the search engines' results have taken place by 9 different users. In this work, only the recall has been evaluated, as all of the approaches including the current search from ARTigo might achieve good precision. Therefore, only the results that do not contain any of the query terms are listed. The question that the users answered while evaluating each method was: how many of the 12 resulted documents are strongly related to the search query? With an overall accuracy of 49.46%, word2vec's vector averaging approach has outperformed synonym lists (46.60%) and doc2vec (19.60%).

Future work

This work leaves many possible further experiments and improvements that can be performed within ARTigo's taggings and search engine. In the aspect of word embeddings, one can re-execute the experiments with word2vec on its other algorithm, skip-gram. Also, the hyper-parameters of word2vec can all be tested again for doc2vec. In order to capture polysemy in word and document vectors, POS-tagging could be applied. However, running automated POS-taggers on unstructured data such as folksonomies would probably not deliver good results. Hence, such a task might be best done by humans and can be very exhaustive depending on how ambiguous the dataset is. Therefore, a GWAP on ARTigo platform could be developed for solving the POS-tagging problem.

Going beyond the well-known synonyms and the grammatically correct related words, one can expand the scope of ARTigo's synonymy by mining morphological variations that could appear in many German dialects. Since a folksonomy is created by a regular user, it might absorb user-related context. The need of filtering out this colloquial content depends on the task being requested. One can even discover common out-of-dictionary terms that can make a lot of sense for many users within a search.

For the sake of ARTigo's semantic quality, other procedures can be executed. Regardless of colloquial words being in the middle of it, problems such as spelling errors and typos can considerably reduce the accuracy of word embedding models as well as of regular TF-IDF-based search. Although ARTigo's game environment takes into account only tags with frequency higher than a certain threshold, typos such as *amnn* (from *mann*) appear in the training set. These tasks of correcting words can also be very time-consuming and at least would require human-based validation for quality assurance.

In the aspect of ARTigo's search functionalities, a technique called relevance feedback could be a good filtering option for refining search queries. Since many ARTigo users have already played or regularly play the GWAPs to tag the artworks, with relevance feedback one can expand the user experience in the search results as well. After entering a simple query, the user selects relevant artworks and gets a refined result set that is similar to the selected artworks. This feature can be easily integrated with the Rocchio algorithm whose implementation is based on TF-IDF weighting and is offered by a Solr plugin[1]. Meanwhile, word embedding models such as word2vec and doc2vec can also be used for finding similar documents given a pre-selected set of documents. To retrieve artworks also by measuring similarity based on other features like color and edges, one could transform the results achieved by Tänzel [23] into labels for a doc2vec model.

ARTigo's current search engine does not support synonym expansion at query time. It means, if a user enters a query composed only by terms that are not in the search engine's term-document matrix or document vectors, the user gets no results back, even if synonyms of these terms are present in the vocabulary. This issue can be addressed by word embeddings as well. To achieve good word similarities between the ones present and the ones missing in the vocabulary, a word2vec model can be trained on a corpus optimized for this task. The most important requirement would be the vocabulary coverage of the corpus to be trained regarding the search engine's corpus. In the best case, ARTigo's vocabulary should be a subset of the expansion vocabulary. One simple solution for the vocabulary problem would be to take a pre-trained word2vec model trained on Wikipedia's dataset. That way, no matter for what terms a user finds results on Wikipedia, it will also find context-related results for those terms on ARTigo. Another meaningful suggestion would be to investigate Elasticsearch[2] as an alternative option to Apache Solr as underlying framework.

Besides the algorithms, a search engine evaluation depends a lot on the chosen test queries. So that the quality of the queries can be ensured, one could evaluate them as well using Fleiss' Kappa, which is a statistical measure for the reliability of agreement between a fixed number of raters. One significant reason for evaluating also the quality of chosen test queries is that every user can interpret them differently. To prove that, two subjects have been observed while selecting the relevant results of the queries for the reported evaluation. The first is a computational linguist, who has not participated in the development of art2vec but uses search engines daily. The second is a specialist on art history and literature. In the query *mensch* (*human being, person*), the first subject has chosen all the images

[1]Solr plugin for relevance feedback by Dice: `https://github.com/DiceTechJobs/RelevancyFeedback`
[2]Elastic's search engine API: `https://www.elastic.co/`

on which a human-like figure is present (8 out of 12). Being more capable of identifying the semantics of the whole result set, without any advising, the second subject has carefully distinguished between persons (1 out of 12), gods and human-like creatures from the mythology for the same result set evaluated by the first subject. That is an impressive difference between the interpretations of the observed subjects during evaluation of such a simple and common search query.

Conclusion

Building a semantic search engine based on word embedding techniques such as word2vec and doc2vec requires the observation of many aspects. These can be related whether to information retrieval or to natural language processing tasks. The quality of the results in all of these tasks has to be evaluated. This thesis has evaluated the semantic expressiveness of the generated word and document vectors. Along with that, the use of these document vectors as retrieval data source has been evaluated by measuring the recall. The experiments in this work have shown that implementing a semantic search with word embeddings can be a helpful way of bringing those documents, previously recognized as false negatives, into the search result set.

Bibliography

[1] Edgar Altszyler, Mariano Sigman, Sidarta Ribeiro, and Diego Fernández Slezak, *Comparative study of lsa vs word2vec embeddings in small corpora: a case study in dreams database*, Master's thesis, 2017, p. arXiv:1610.01520.

[2] Marco Baroni, Georgiana Dinu, and German Kruszewski, *Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors*, Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (2014), 238–247.

[3] Daniel Cera, Yinfei Yanga, Sheng yi Konga, Nan Huaa, Nicole Limtiacob, Rhomni St. Johna, Noah Constanta, Mario Guajardo-Cespedes, Steve Yuanc, Chris Tara, Yun-Hsuan Sunga, Brian Stropea, and Ray Kurzweila, *Universal sentence encoder*, arXiv e-prints (2018), arXiv:1803.11175.

[4] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa, *Natural language processing (almost) from scratch*, Journal of Machine Learning Research 12 (2011), 2493–2537.

[5] Oliver Deck, Tobias Eder, and Jaderson R. Webler, *Multi-layered classification for named entity recognition on the KONVENS 2014 dataset*, Seminar paper, Center for Information and Language Processing (CIS), Ludwig Maximilian University of Munich, 2016, Available on: `https://jumpshare.com/v/57f5Qj2TJvKhmQ8i4PAJ`.

[6] Wenpeng Hu, Jiajun Zhang, and Nan Zheng, *Different contexts lead to different word embeddings*, Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers (2016), 11–17.

[7] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov, *Bag of tricks for efficient text classification*, Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics **2** (2017), arXiv:1607.01759.

[8] Quoc Le and Tomas Mikolov, *Distributed representations of sentences and documents*, Proceedings of the 31st International Conference on Machine Learning **32** (2014), arXiv:1405.4053.

[9] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze, *An introduction to information retrieval*, Cambridge University Press, 2009.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, *Efficient Estimation of Word Representations in Vector Space*, arXiv e-prints (2013), arXiv:1301.3781.

[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean, *Distributed representations of words and phrases and their compositionality*, arXiv e-prints (2013), arXiv:1310.4546.

[12] Eneko Pinzolas Murua, *Word embeddings in search engines, quality evaluation*, 2017, Available on: `http://ad-publications.informatik.uni-freiburg.de/theses/Bachelor_Eneko_Pinzolas_2017.pdf`.

[13] Andreas Müller, *Analyse von Wort-Vektoren deutscher Textkorpora*, Bachelor thesis, Institute of Software Engineering and Theoretical Computer Science, TU Berlin, 2015.

[14] Thomas Niebler, Luzian Hahn, and Andreas Hotho, *Learning word embeddings from tagging data: A methodological comparison*, Proceedings of the LWDA 2017 Workshops: KDML, FGWM, IR, and FGDB (2017).

[15] Jeffrey Pennington, Richard Socher, and Christopher D. Manning, *Glove: Global vectors for word representation*, Master's thesis, Computer Science Department, Stanford University, 2014.

[16] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena, *Deepwalk: online learning of social representations*, Conference on Knowledge Discovery and Data Mining - KDD'2014 (2014).

[17] Iman Saleh and Neamat El-Tazi, *Finding semantic relationships in folksonomies*, Research paper, Faculty of Computers and Information, Cairo University, 2018.

[18] Gerard Salton, Anita Wong, and Chung-Shu Yang, *A vector space model for automatic indexing*, Communications of the ACM **18** (1975), no. 11, 613–620.

[19] Tobias Schnabel, David Mimno, and Thorsten Joachims, *Evaluation methods for unsupervised word embeddings*, 2015, pp. 298–307.

[20] Christina Schneider, *Merkmalsbasiertes Opinion Mining anhand von Produktbewertungen in Onlineportalen*, Master thesis, Hochschule der Medien Stuttgart, 2015.

[21] Hinrich Schütze and Jan O. Pedersen, *A co-occurrence-based thesaurus and two applications to information retrieval*, Information Processing & Management **33** (1997), 307–318.

[22] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei, *Line: Large-scale information network embedding*, International World Wide Web Conference WWW. 2015 (2015).

[23] Tobias Tänzel, *Measuring similarity of artworks using multidimensional data*, Master's thesis, Institute of Informatics, Ludwig Maximilian University of Munich, 2017.

[24] Thomas Vander Wal, *Explaining and showing broad and narrow folksonomies*, 2005, Available on: `http://www.vanderwal.net/random/entrysel.php?blog=1635`, Last accessed: 2019-02-01.

[25] Christoph Wieser, *Building a semantic search engine with games and crowdsourcing*, Ph.D. thesis, Institute of Informatics, Ludwig Maximilian University of Munich, 2014.

[26] Christoph Wieser, François Bry, Alexandre Bérard, and Richard Lagrange, *ARTigo: Building an artwork search engine with games and higher-order latent semantic analysis*, First AAAI Conference on Human Computation and Crowdsourcing (2013).

APPENDIX A

Art2vec's tool set

0. Before starting

- the art2vec repository can be found at:
  `https://gitlab.pms.ifi.lmu.de/bognerm/art2vec`
- clone this repository into the webserver root, where python and php files are able to run
- open a terminal to work with the tools, change the directory to this repository and stay there to run the tools
- every tool references files from this directory on, so if the current directory is not art2vec then the tools will not work
- all the tools's arguments are required and described below
- one can run every tool separately or just run `bash run.sh` to automatically run all the tools one by one with the already optimized argument values for the ARTigo dataset
- all file names for word2vec models are identified by:
  `preprocess_algorithm_optimization_min-count_alpha_window_negative_vector-size`
- all file names for doc2vec models are identified by:
  `preprocess_alg_opt_wv-average_min-count_alpha_window_negative_vector-size`

1. Installing solr

- If needed, run the tool `bash solr/tool/dependencies.sh` before installing Solr. This tool installs: java, apache2, curl, php, python3, gensim, sklearn, matplotlib, numpy.
- **ATTENTION!!! The next tool will uninstall any existing Solr version in the system! All data in Solr will be lost!**
- This tool removes exiting Solr versions, installs the proper Solr version for this project and creates and configures Solr core.
- **$** `bash solr/tool/install.sh`

- In case an error occurs due to missing dependencies, just rerun these both tools.
- Check if the installation succeeded: `http://localhost:8983/solr/#/art2vec`

2. Pre-processing tags

- All the data sources for this project are stored in `data/sources` as .csv files containing different information about the artworks. The goal of this tool is to pre-process the taggings as desired and stored them in a proper format for later word2vec/doc2vec training and merging with other data fields.
- IMPORTANT: replace 1 with 0 "to turn of" the arguments 5-8.
- **$** `/usr/bin/python3 data/tool/preprocess.py artigo-taggings.csv 0 1 2 1 30 1 1`
- Argument 1: tagging file name containing at least columns that represent document id, tag and tag count
- Argument 2, 3, 4: column position of document id, column position of tag, column position of tag count
- This tool takes for every document all its taggings and merges them into a single lined text.
- eg: **mann 3 — frau 2 — bier 1**
- Argument 5: repeat tags according to tag count
  - *mann mann mann frau frau bier*
- Argument 6: max number of repetitions for each tag, if 0 use tag count, e.g. 2:
  - *mann mann frau frau bier*
- Argument 7: shuffle tag order (outer)
  - *frau frau bier mann mann*
- Argument 8: shuffle tag order (inner)
  - *mann bier frau mann frau*

3. Training word2vec

- This tool will use the tags preprocessed in the section 2 as corpus to train a word2vec model and store it in `word2vec/models`.
- **$** `/usr/bin/python3 word2vec/tool/train.py 13011.gz 0 0 5 0.025 5 5 100`
- Arguments: file name, algorithm (1 for skip-gram, 0 for cbow), optimization (1 for negative sampling, 0 for hierarchical softmax), min count, training rate, window, negative samples, vector size

4. Evaluating word2vec models

- An adaptation of `https://github.com/devmount/GermanWordEmbeddings/blob/master/evaluation.py`
- **$** `/usr/bin/python3 word2vec/tool/evaluate.py 13011_0_0_150_003_25_15_150`

- Argument 1: model name
- The evaluation statistics is saved under `word2vec/data/results/`.

5. Averaging word2vec models

- This tool averages the word vectors from a given word2vec model using sum or mean into document vectors.
- **$** `/usr/bin/python3 word2vec/tool/average.py 13011_0_0_150_003_25_15_150 1`
- Argument 1, 2: model name, average (1: mean, 0: sum)

6. Extracting document vectors from word2vec models

- This tool loads the given word2vec model, extracts its vectors and writes them in a .csv file for merging with the tags.
- **$** `/usr/bin/python3 word2vec/tool/extract.py 13011_0_0_150_003_25_15_150_1.npy w2v_150`
- Arguments 1, 2: doc2vec model, csv column name (must be same as in `solr/configs/managed-schema`)

7. Training doc2vec

- This tool will use the tags preprocessed in the part 2 of this manual as corpus to train a doc2vec model and store it in `doc2vec/models`.
- **$** `/usr/bin/python3 doc2vec/tool/train.py 13011.gz 1 0 0 150 0.030 25 15 150`
- Arguments 1, 2, 3: data file name (gzip format), vector size, iterations

8. Evaluating doc2vec models

- It evaluates a given number of documents in 3 different categories based on pre-defined corpus-specific search queries. To changed these queries, please edit this tool.
- **$** `/usr/bin/python3 doc2vec/tool/evaluate.py 13011_1_0_0_150_003_25_15_150 100 10`
- Argument 1, 2, 3: model name, number of documents to be sampled for each of the 3 categories, number of most-median-least similar documents for each of the samples to be examined
- The evaluation statistics is saved under `doc2vec/data/results/`.

9. Extracting document vectors

- This tool loads the given doc2vec model, extracts its vectors and writes them in a .csv file for merging with the tags.
- **$** `/usr/bin/python3 doc2vec/tool/extract.py 13011_1_0_0_150_003_25_15_150 d2v_150`
- Arguments 1, 2: doc2vec model, csv column name (must be same as in `solr/configs/managed-schema`)

10. Merging tags and metadata with word2vec and do2vec document vectors

- Builds the final file for importing into Solr.
- **$** `bash data/tool/merge-all.sh`

11. Import vector file

- After merging the final file, run this tool to import all the data at once into Solr.
- **$** `bash solr/tool/import.sh`
- ATTENTION!!! This tool uses **curl** which can break with message ”empty reply from server”.
- Normally solr server continues to import after this message. If this does not happen, then run `bash solr/tool/install.sh` and `bash solr/tool/import.sh` again.

12. Art2vec's frontend

- open this art2vec repository in the browser and one can start making searches
- URL may be: `http://localhost/art2vec/`

EXTRA TOOLS

---

13. Query vector generation for word2vec models

- It expands the query terms by appending the 3 most similar word of each term. Then it averages (mean) all word vectors contained in the expanded query and so building a query vector.
- Arguments 1, 2: full path to word2vec model (IMPORTANT: not averaged!), ”query terms in quotes”
- **$**
```
/usr/bin/python3 word2vec/tool/query.py
/full/path/to/13011_0_0_150_003_25_15_150 "mann frau bier"
```
- *-0.075648836,0.62597,1.39151555,-1.21589438368,0.14489697,0.525327,0.387357,0.6740211,1.7528240,0.44816,0.09120425,-2.19873714,-0.9967,-0.839131,...*

15. Query vector generation for doc2vec models

- It expands the query terms by appending the 3 most similar word of each term. Then it repeats every term based on its average frequency in the corpus. Finally these expanded terms are used for inferring a document vector from the pre-trained doc2vec model, which is then the query vector.
- Arguments 1, 2, 3, 4: full path to doc2vec model, ”query terms in quotes”, expand with most similar (1) or not (0), repeat each term based on its document frequency (1) or not (0), generate query vector through infer vector function of doc2vec (0) or retrieve most similar documents from the inferred document vector (1)
- **$**
```
/usr/bin/python3 doc2vec/tool/query.py 13011_1_0_0_150_003_25_15_150 "mann
frau bier" 1 1 0
```

- *0.10199931,-0.08667883,0.0072245025,-0.025322046,0.07703773,-0.019855263,0.106641,0.026173364,-0.07345875,0.10025604,0.038576134,-0.06285023,-0.09113559,0.047214482,0.013344384,0.07479666,...*

16. Synonym file generation

- Needs to be run with super user. It creates a Solr-friendly synonym file which is then written to `/var/solr/data/art2vec/conf/synonyms.txt`.
- No further configuration is needed for Solr to index the synonyms.
- Arguments 1, 2, 3: model file name, top n terms to build lists, top n most similar word for each term
- **$** `sudo /usr/bin/python3 word2vec/tool/synonym.py 13011_0_0_150_003_25_15_150 500 3` `sudo service solr restart`

17. Bulk training for word2vec

- This tool helps on experimenting parameter values automatically.
- Arguments 1, 2, 3, word2vec args, LAST (optional): word2vec parameter name to be experimented, algorithm, optimization, all word2vec parameter values to be used including the one to be experimented (fill all with zeros to use the defaults), pre-processed tag file
- **$** `/usr/bin/python3 word2vec/tool/experiment.py min_count 0 0 0 0 0 0 0 13011.gz`
- Parameter ranges to be experimented:
  - `pre_process` = ['0000.gz', '0010.gz', '0011.gz', '1000.gz', '1010.gz', '1011.gz', '12011.gz', '13011.gz', '13511.gz']
  - `min_count` = [1, 5, 15, 30, 50, 75, 100, 150, 175, 200]
  - `alpha` = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.010]
  - `window` = [3, 5, 10, 15, 20, 25]
  - `negative` = [2, 3, 5, 10, 15, 20, 25]
  - `vector_size` = [50, 100, 150, 200, 300, 500]
- All the trained models are saved in the path word2vec/models. The evaluation tool is executed when the training of each model is finished.

18. Bulk training for doc2vec

- This tool helps on experimenting parameter values automatically.
- Arguments 1, 2, 3, 4, word2vec args, LAST (optional): word2vec parameter name to be experimented, algorithm, optimization, averaging, all word2vec parameter values to be used including the one to be experimented (fill all with zeros to use the defaults), pre-processed tag file
- **$** `/usr/bin/python3 word2vec/tool/experiment.py min_count 1 0 0 0 0 0 0 0 13011.gz`
- Parameter ranges to be experimented: same as above for word2vec.
- All the trained models are saved in the path doc2vec/models.

19. **Search engine's recall evaluation**

    - Building query files
        - write the search queries to be evaluated in the file `eval/queries.txt` separated by new line.
        - run build file: `php eval/build.php`
        - all the results for each query for every method (synonym, word2vec and doc2vec) are then saved into `eval/var/queries/`.
    - Opening eval frontend page:
        - URL may be: `http://localhost/art2vec/eval/`
        - evaluation real-time statistics: `http://localhost/art2vec/eval/rslts/`

20. **Extracting vocabulary**

    - An adaptation of
      `https://github.com/devmount/GermanWordEmbeddings/blob/master/vocabulary.py`
    - It creates a text file containing the entire vocabulary with its frequency count from the given model.
    - Arguments 1, 2: model name, file to store the vocabulary
    - **$** `/usr/bin/python3 word2vec/tool/vocabulary.py 13011_0_0_150_003_25_15_150 word2vec/data/13011_0_0_150_003_25_15_150.vocab`

21. **Visualizing word relationships**

    - An adaptation of
      `https://github.com/devmount/GermanWordEmbeddings/blob/master/visualize.py`
    - This tool plots the position of word pairs in the vector space using PCA. To change the pairs to be plotted, please edit this file.
    - Arguments 1, 2: model name, file to store the vocabulary
    - **$** `/usr/bin/python3 word2vec/tool/visualize.py 13011_0_0_150_003_25_15_150 word2vec/data/13011_0_0_150_003_25_15_150.vocab`

22. **Solr request URLs examples**

    - `SOLR_URL = 'http://localhost:8983/solr/#'`
    - word2vec:
      `SOLR_URL/art2vec/query?fl=ids&q={!vp%20f=w2v_150%20vector='-0.075643,,...'}`
    - doc2vec:
      `SOLR_URL/art2vec/query?fl=ids&q={!vp%20f=d2v_150%20vector='0.101999,...'}`
    - standard keyword request (TF-IDF):
      `SOLR_URL/art2vec/select?df=tags&fl=id,tags&q='mann%20frau%20bier'`
    - standard keyword request with excluded keywords (TF-IDF):
      `SOLR_URL/art2vec/select?df=tags&fl=id,tags&q='bart'&fq=-'mann'`
    - relevance feedback request (Rocchio algorithm):
      `SOLR_URL/art2vec/rf?rf.q=id:5715+id:11750&df=tags&fl=id,tags`
    - standard by id request: `SOLR_URL/art2vec/get?ids=334380,304976&fl=id,tags`