TECHNISCHE UNIVERSITÄT DRESDEN

Word Embeddings in Database Systems

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Dipl.-Inf. Michael Günther
geboren am 06. Mai 1994 in Dresden

Gutachter:

Prof. Dr.-Ing. Wolfgang Lehner
Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur
Lehrstuhl für Datenbanken
01062 Dresden

Prof. Dr. Christian Bizer
Universität Mannheim
Fakultät für Wirtschaftsinformatik und Wirtschaftsmathematik
Institut für Informatik und Wirtschaftsinformatik
Lehrstuhl für Informationssysteme V: Webbasierte Systeme
B6, 26
68159 Mannheim

Tag der Verteidigung: 9. November 2021

ABSTRACT

Research in natural language processing (NLP) focuses recently on the development of learned language models called word embedding models like word2vec [MSC+13], fastText [BGJM17], and BERT [DCLT19]. Pre-trained on large amounts of unstructured text in natural language, those embedding models constitute a rich source of common knowledge in the domain of the text used for the training. In the NLP community, significant improvements are achieved by using those models together with deep neural network models. To support applications to benefit from word embeddings, we extend the capabilities of traditional relational database systems, which are still by far the most common DBMSs but only provide limited text analysis features. Therefore, we implement *(a) novel database operations involving embedding representations* to allow a database user to exploit the knowledge encoded in word embedding models for advanced text analysis operations. The integration of those operations into database query language enables users to construct queries using novel word embedding operations in conjunction with traditional query capabilities of SQL. To allow efficient retrieval of embedding representations and fast execution of the operations, we implement *(b) novel search algorithms and index structures for approximated kNN-Joins* and integrate those into a relational database management system. Moreover, we investigate techniques to optimize embedding representations of text values in database systems. Therefore, we design *(c) a novel context adaptation algorithm*. This algorithm utilizes the structured data present in the database to enrich the embedding representations of text values to model their context-specific semantic in the database. Besides, we provide *(d) support for selecting a word embedding model* suitable for a user's application. Therefore, we developed a data processing pipeline to construct a dataset for domain-specific word embedding evaluation. Finally, we propose *(e) novel embedding techniques for pre-training on tabular data* to support applications working with text values in tables. Our proposed embedding techniques model semantic relations arising from the alignment of words in tabular layouts that can only hardly be derived from text documents, e.g., relations between table schema and table body. In this way, many applications, which either employ embeddings in supervised machine learning models, e.g., to classify cells in spreadsheets, or through the application of arithmetic operations, e.g., table discovery applications [NZPM18], can profit from the proposed embedding techniques.

CONTENTS

ACKNOWLEDGMENTS

I am very grateful for the opportunity to work on this dissertation project and all the support I got in the last three and a half years. Thus, I want to thank Wolfgang Lehner for making this possible and for his trust in me. Furthermore, I would like to thank Maik Thiele, who believed in me and convinced by to apply for the Ph.D position instead of an internship. I am grateful for all the knowledge he shared with me and this competent feedback to all my ideas, even if I changed my mind and reconsidered my approaches three times a week. Moreover, I also want to thank Christian Bizer for his commitment to creating the external review.

I want to thank all my current and former colleagues at the chair, who make the working atmosphere enjoyable. Moreover, I want to say thanks for all the helpful discussions about my research problems. Further, I want to thank everybody who has organized a KuK or attended my KuKs. Besides, I also want to thank the colleagues form the research training group RoSI. I also want to thank Ulrike and Ines for their support with administrative issues. Furthermore, I want to thank Ulrike for proofreading my publications. In addition, I would also like to thank all the students who contributed to my research.

Finally, I want to thank my family for always supporting me and making my life easier in situations where I was stressed. I want to thank Annika for encouraging me and always paying attention to my doubts and concerns. I also want to thank all my friends who distracted me from time to time from my dissertation so that I could fully enjoy my free time.

Michael Günther
Dresden, September, 2021

1 INTRODUCTION

Research in natural language processing has received substantial progress by using deep learning techniques in recent years. Specifically, significant improvements are achieved by using word embedding models like word2vec [MSC+13, MCCD13], fastText [BGJM17], and BERT [DCLT19]. Those models are pre-trained on large amounts of unstructured text in natural language. This allows them to encode text snippets as vector representations where semantic relations between text values are represented as spatial relations. In this way, embedding models can be employed by downstream applications to interpret input text values and have found application in a wide range of NLP [XFS14, Sie15] and information retrieval tasks [ZKBA15, NMCC16, ZML+20]. The impressive success of word embedding models has also attracted attention in the data management community. Especially in data interpretation [ETJ+18, LLS+20a] and data discovery tasks [FMQ+18, NPZ+20], major successes are achieved by employing word embedding models. However, the text analysis capabilities of the popular database management systems still only support traditional text-processing functions, which are restricted to analyze and transform syntactic properties. Thus, they can not capture semantic relations between words, which are not expressed by syntactic similarities. Accordingly, this thesis aims to improve text analysis in database systems by integrating word embedding functionality in relational database systems.

In the first place, we aim at extending the text processing capabilities with novel functions using the powerful abilities of word embedding techniques. Specifically, we encode text values in the database by dense vectors generated with word embedding models. Machine learning applications can then easily and efficiently access those vectors. In addition, novel word embedding operations largely extend the capabilities of the query language. Therefore, those operations process the embedding representations maintained in the database. To efficiently execute those novel operations, we design novel index structures and search algorithms for operating on high dimensional vectors in the database. Moreover, we developed a RETRO framework to improve word embedding representations by using the semantic context of text values in the database system. The framework is integrated into the database to efficiently generated optimized representations for new text values. ML applications using the improved embeddings can profit from this optimization process. To further support the database user, we investigate word embedding evaluation techniques and construct a dataset for domain-specific embedding model evaluation. By reviewing recent research works in the data management community, we noticed that many of those applications use word embedding models pre-trained on text. Thus, we investigate techniques to pre-train embedding models on tabular data and evaluate their usefulness on table discovery tasks.

1.1 CONTRIBUTION

The contribution of this thesis can be summarized in five main aspects.

1. **Word Embedding Operations:** Based on the powerful abilities of word embedding techniques, we integrate additional functionality for text analysis and machine learning into a PostgreSQL database system. Therefore, we developed a database extension [Gün18, GTLY19] to add novel query capabilities to the SQL query language by additional word embedding operations. Those operations utilize vector representations of the text values in the database generated by word embedding models. Essential operations are similarity quantification, analogy calculation, grouping, clustering, and similarity joins.

2. **Efficient Processing of Learned Representations:** The word embedding representations of text values are stored in the database. To efficiently access those vectors and execute the novel operations described above, we implement novel index structures and a novel search algorithm [GTL19a]. Besides word embedding representations, search applications using other learned representations can also profit from those algorithms.

3. **Optimizing Representaions of Database Text Values:** A naïve application of a word embedding model is not sufficient to represent the meaning of text values in a database which is often more specific than the general semantic encoded in the raw word embedding. This leads to sub-optimal embedding representations of text values and potentially undesired results of word embedding operations when applied to the text values in the database. Thus, to improve the embedding representation, we designed the relational retrofitting algorithm RETRO[GTL20, GOTL20] to utilize the information given by the disposition of the text values in the database schema, e.g., which text values appear in the same column or are related. An additional variant of the algorithm provides a fast method to generate optimized embedding representations for text values added later to the database in an online manner. Our evaluation platform RETROLIVE [GTNL20] support users in the hyperparameter tuning by visualizing the effects on the resulting embedding representations and allows evaluating machine learning models on the optimized embedding representations.

4. **Support for Embedding Model Selection:** Our PostgreSQL database extension provides methods to switch between word embedding models underlying the execution of the embedding operations. In this way, the user can shift between different notions of similarity. However, dependent on the domain of the text values in the database and the application domain, different embedding models are suitable for a specific task. To support users in the decision of a model, the system should provide recommendations based on an extensive domain-specific evaluation of the embedding models. Therefore, we propose an approach [GSTL20b] to automatically construct a facet-structured evaluation dataset from tabular data. The dataset is organized in a facet structure to facilitate a fine-granular evaluation of the performance of the models. For the construction, one can use a large corpus of tables, e.g., a Web table corpus or a collection of a company's CSV files.

5. **Embedding Representation of Schema Information:** Word embedding models pretrained on text turn out to be a poor choice to model semantic relations between schema information in the column title and instance data appearing in the table body. Moreover, text values in database systems are differently interpreted than

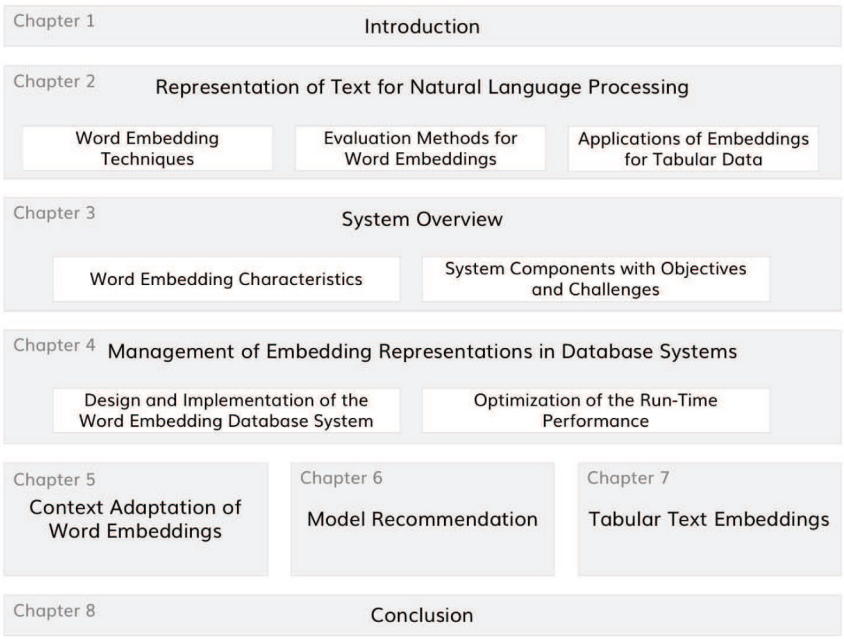


Figure 1.1: Structure of this Thesis

words and phrases in documents. Thus, when processing tabular data word embedding model trained on documents might be suboptimal. To overcome this limitation, we developed a novel embedding technique for text values in tables compatible with our system [GTGL21]. Our embedding approach can effectively model relations between schema and instance terms. In an extensive evaluation, we pretrain embeddings on Web tables and demonstrate that it can be effectively applied to text values in various table formats, e.g., Web tables, spreadsheets, CSV files of open data repositories.

1.2 OUTLINE

Figure 1.1 presents the organization of this thesis. In Chapter 2, we describe the fundamentals for this thesis. Specifically, we introduce common word embedding techniques and their properties. Moreover, we introduce methods for evaluating word embedding models and applications of word embeddings for data in tables. In Chapter 3, we present a detailed overview of the research questions we address. Therefore, we analyze popular pre-trained word embedding models, which we use in the following chapters of the thesis. Afterward, we introduce the main components of the integration of word embedding techniques into database systems. Thereby, we define requirements for all those components, which are addressed by our research presented in the following chapters. Chapter 4 presents the design and the implementation of our word embedding database systems. This also includes the integration of novel embedding operations. Moreover, we tackle performance issues of word embedding operations. Therefore, we propose a novel algorithm for approximated kNN-Joins in this chapter. In the following three chapters, we present additional system components which address the research problems described in Chapter 3. We start with our relational retrofitting framework in Chapter 5. Besides, this chapter also describes an associated Web interface and an online version of the algorithm to support efficient optimization of updated text values. In Chapter 6,

we present an algorithm for constructing a dataset for domain-specific word embedding evaluation. Moreover, we use it to generate a comprehensive dataset and employ it in an extensive evaluation of common word embedding models. Chapter 7 presents novel models for training word embeddings on text in tables. In an extensive evaluation, we show the usefulness of our models for different tasks and table formats. Finally, we conclude in Chapter 8. Thereby, we summarize the contribution and results and suggest some directions for future work.

2

REPRESENTATION OF TEXT FOR NATURAL LANGUAGE PROCESSING

To assess the potential of word embedding methods for database systems, we give a short introduction to common practices in state-of-the-art natural language processing systems and how word embeddings are used by those systems in Section 2.1. Then, we take a closer look at word embedding methods and related embedding methods in Section 2.2. This is followed by Section 2.3 where we shortly discuss how word embedding models are evaluated. Finally, we discuss properties of word embeddings that can be utilized for processing textual data in tables and look at applications that build upon those in Section 2.4.

2.1 NATURAL LANGUAGE PROCESSING SYSTEMS

Natural language is a tool humans regularly use to communicate with each other. Thus, there are large amounts of textual data in natural language which are valuable for analyzing them with computers. Moreover, in many applications, it is desired to implement interfaces where users can communicate to the system in natural language and/or that the system can communicate its output in natural language (e.g. to increase their usability). All these require techniques to make human languages accessible to computers which are the subject of the field of natural language processing (NLP) [Eis19]. Understanding and producing natural language is a challenging task. Even humans able to speak a language are unable to formally define it. Thus, for many NLP tasks commonly used tools rely on supervised machine learning techniques [Gol17]. Formerly, linear classification models have been widely used in NLP. However, neural networks, and especially deep learning techniques, got more prevalent more recently [Man15, YHPC18]. Those deep learning models implement complex classification functions by learning hierarchies of representations from the input. In [Eis19] their dominance is explained by rapid advantages in the deep learning community, the compatibility of such models with word embeddings (see Section 2.2), and their efficient implementation on GPUs and TPUs leading to better hardware utilization.

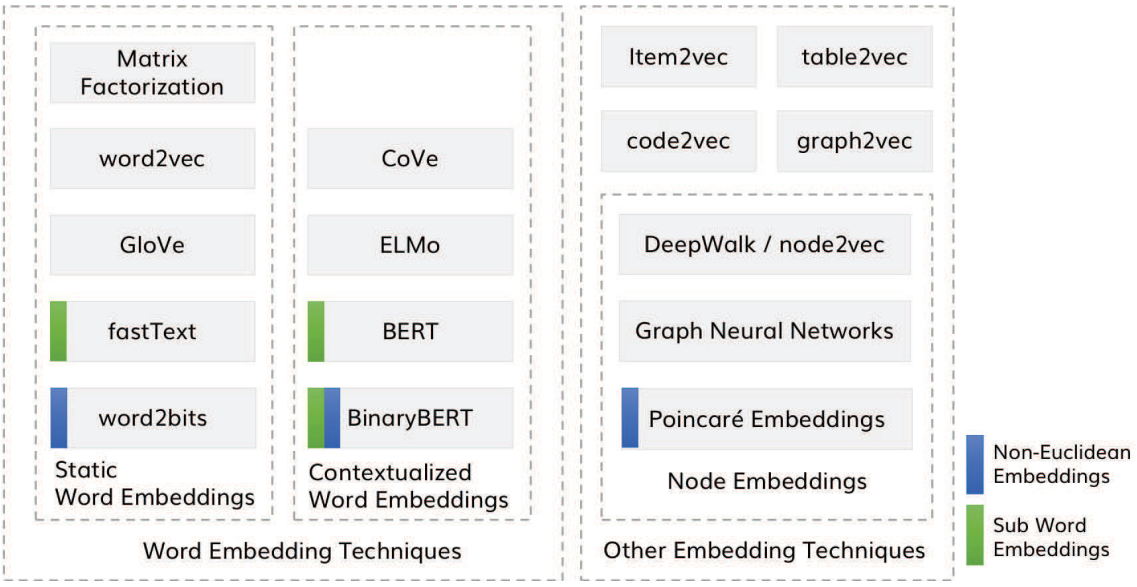


Figure 2.1: Variety of Embedding Techniques

NLP Pipelines The rise of deep learning in NLP has also changed the structure of typical pipelines for NLP tasks. Today, supervised NLP tasks are typically performed by end-to-end neural networks [BG19]. Most of those networks utilize so-called word embedding models to encode terms into continuous vector representations. One can distinguish between static (see Section 2.2.1 and Section 2.2.2) and contextualized word embeddings (see Section 2.2.3). Contextualized word embedding models do not statically map a word to a vector, but also consider other terms in the sentence where the word occurs. Word embedding vectors can either serve as input vectors or as parameters of the neural network model which are fine-tuned during the training. While fine-tuning can increase the performance of a model, it can also hurt the generalization properties of the neural network, because only embeddings observed in the training data can be adjusted [Gol17]. Here, fine-tuning is very common.

2.2 WORD EMBEDDING MODELS

The objective of building word embedding models is to encode the semantic of words into a numerical representation. This objective has a long history. Gottfried Wilhelm Leibniz already tried to use prime numbers to implement a calculus [Lei76] that assigns words to numbers from which semantic relations should be derived by arithmetic relations [Mit79].

Word embedding models perform a transformation from terms of a vocabulary V into a continuous vector space $\mathbb{R}^d$. A straightforward vector encoding for terms is the one-hot encoding where the dimensionality d corresponds to the vocabulary size $|V|$ and each term is represented by the vector that holds a 1 at the index of the term in the vocabulary and zeros everywhere else. In contrast, word embedding models rather assign terms to dense vectors of much lower dimensionality. The one-hot-encoding can be seen as a *local representation* since each element of a vector encodes the presence of a specific term. State-of-the-art embeddings are so-called *distributed representations* [HMR86]. Here, terms are not assigned to specific dimensions but rather by a d dimensional vector specific for this term. Those vectors can be seen as activation patterns spanning over all d dimensions.

Similarly, in the human brain a representation of an object is not encoded by one neuron but rather by activation statuses of several neurons. Those word vectors are learned by a word embedding algorithm which is applied to a large corpus of text.

According to the *Distributional Hypothesis*, the semantic of words can be derived from their distribution in texts. Thereby, words occurring in the same contexts usually have a similar meaning [Har54]. Based on this hypothesis, word embedding models capture distributional properties of words in dense vector representations.

Types of Embedding Techniques: There is a large variety of different word embedding models (see Figure 2.1). We distinguished them into static and contextualized word embedding techniques. Early static methods utilize *matrix factorization* to generate embeddings (Section 2.2.1). Later techniques generating *learned distributed representations* (Section 2.2.2) where developed, e.g., word2vec [MSC+13, MCCD13] and GloVe [PSM14]. In Section 2.2.3, we introduce the recently developed *contextualized word embedding models*, which differ from the other methods in the sense that the term to vector mapping is performed dependent on the surrounding words in the sentence. Ordinary embedding techniques map terms directly to vectors. To represent unknown terms, several techniques model subwords, e.g., n-grams, as vectors to enable users to embed tokens not observed in the text corpus by the embedding training algorithm. Besides word embedding models, there is a wide range of methods mapping other entities into a dense vector space, e.g., item2vec [BK16], graph2vec [NCV+17], code2vec [AZLY19], and table2vec [ZZB19]. Especially for embedding nodes in graphs, a large variety of techniques have been proposed [GF18]. Since those techniques have a wide range of applications, including data management tasks, we take a closer look at them in Section 2.2.6. Traditionally embedding techniques produce representations in a Euclidean vector space. However, for some applications, other non-Euclidean spaces are favorable, which leads to the development of embedding techniques like word2bits [Lam18], BinaryBERT [BZH+21], and Poincaré embeddings [NK17] (see Section 2.2.7).

2.2.1 Matrix Factorization Methods

Many word embedding methods rely on term-context matrices $M = [m_{i,j}] \in \mathbb{R}^{|V| \times |C|}$ where each row i represents a word and each column j represents a linguistic context. Each entry $m_{i,j}$ is either zero or represents a weight if word i is present in context j. Different definitions of contexts $c \in C$ are possible. For large document collections, each document can be considered as a context [DDF+90]. The weight can then be defined by the frequency of the word in the document. Also popular is the *sliding window* approach. Thereby sequences of $2w + 1$ words are observed by shifting a window of this size over the sentences in a document. For the center words in those sequences, the surrounding words act as contexts. Here, the weight can be defined as the number of sequences a center and context word co-occur. Moreover, the distance between a center word and a context word can also be used for the weighting [PSM14]. Both context types, word frequency as well as the number of co-occurrences favor associations between frequent terms and contexts. To reduce this bias, the *point-wise mutual information* [CH90] can be used as an alternative association measure:

$$PMI(w, c) = \log \frac{P(w, c)}{P(w)P(c)} \tag{2.1}$$

Here, $P(w)$ and $P(c)$ refer to the probabilities of observing the word w and context c, and the term $P(w, c)$ denotes the joint probability of observing w in context c. Further

considerations of the use of PMI for creating term-context matrices can be found in [PL02, TP10].

To reduce the dimensionality of the term-context matrix *Singular Value Decomposition* is used. If the dimensionality of the columns of M is reduced to obtain a dense vector for each context, e.g., for document retrieval, this is called *Latent Semantic Indexing* [DDF+90]. When the dimensionality of the rows is reduced to obtain dense vectors for terms, e.g., to reveal word similarity, this is called *Latent Semantic Analysis* [LD97]. SVD decomposes the matrix M into a matrix product $U\Sigma V^T$ such that U and V are in column orthonormal form (columns are orthogonal and have unit length), and Σ is a diagonal matrix of singular values. Afterward, Σ_k can be obtained by selecting the top k singular values. Moreover, U_k and V_k are obtained by selecting only the k leftmost columns of U and V. Then, $M' = U_k\Sigma_k V_k^T$ constitutes a low-rank approximation of M. Moreover, a matrix of k-dimensional embeddings $E = U_k\Sigma_k$ can be calculated. Here, the rows are a low dimensional approximation of the high dimensional rows of M in the sense that the dot product of two rows in E is equivalent to the dot product of two rows M' [Gol17]. In practice, U_k is sometimes directly used as the embedding matrix. The embedding vectors can then be used to quantify similarity between words [LD97]. Usually, a dimensionality between 50 and 300 is selected [Gol17]. The dimensionality reduction has several advantages: On the one hand, the term-context matrix is usually very sparse and can contain millions of contexts, making it very unhandy. On the other hand, those large vectors might be noisy, especially for infrequent terms and contexts. By lowering the dimensionality, the noise can be reduced. Moreover, by limiting the dimensionality, the latent meaning of words can be revealed and similarity measures can be improved [TP10].

2.2.2 Learned Distributed Representations

Besides representations directly generated from a co-occurrence matrix, several methods initialize word vectors randomly and modify them by an optimization algorithm to satisfy an objective function. Thereby, a word is represented by a d-dimensional vector ($d << |V|$). However, a dimension is not interpretable like a column in the term-context matrix introduced in Section 2.2.1. A property of a word is rather captured by a combination of values of multiple dimensions. The most popular method in this category is the word2vec technique proposed in [MSC+13, MCCD13]. Here, two models are introduced: *Continuous Bag of Words (CBOW)* and *Skip-Gram*. Both models use a neural network to model probabilistic relations between a center word and context words. Figure 2.2 depicts the architectures of the models. Both models use a sliding window approach to obtain sequences of words $w_{t-l}, \ldots, w_t, \ldots, w_{t+l}$ from a large corpus with a fixed-length $k = 2l + 1$. The CBOW model tries to predict the center word w_t from the surrounding context words. In contrast, the Skip-Gram model tries to predict the surrounding words given the center word. Both models consist of two embedding matrices W and W'. The matrix W also serves the matrix of the actual word embeddings after the model training is finished, however, one might also use W' or a combination of both matrices [NMCC16]. Before the training, both matrices are initialized randomly. The models represent each word in form of a sparse one-hot encoding $\mathbf{x_i}$. Those sparse vectors are then transformed into dense d-dimensional vector representations by multiplying them with the matrix W. While in the Skip-Gram model $\mathbf{h} = W\mathbf{x}$ is directly obtained from the matrix, in the CBOW model $\mathbf{h} = \frac{1}{2l}(\mathbf{x_1} + \ldots + \mathbf{x_k})$ is obtained by averaging the vectors of the context words. Then $\mathbf{h}$ is transformed back into a $|V|$-dimensional vector of scores by multiplication with W'. A straight-forward approach to predict probabilities for the center word in the CBOW model and a context word in the Skip-Gram model is to apply a softmax function $\sigma(\mathbf{y})_i = \frac{exp(y_i)}{\sum_{j=1}^{|V|} exp(y_j)}$ on the output layer. However, the training of such a model is too inefficient for a large collection of text [Ron14]. To solve this problem, [MSC+13] propose to use a technique called negative sampling. Alternatively, one can use hierarchical softmax as described in [MCCD13].

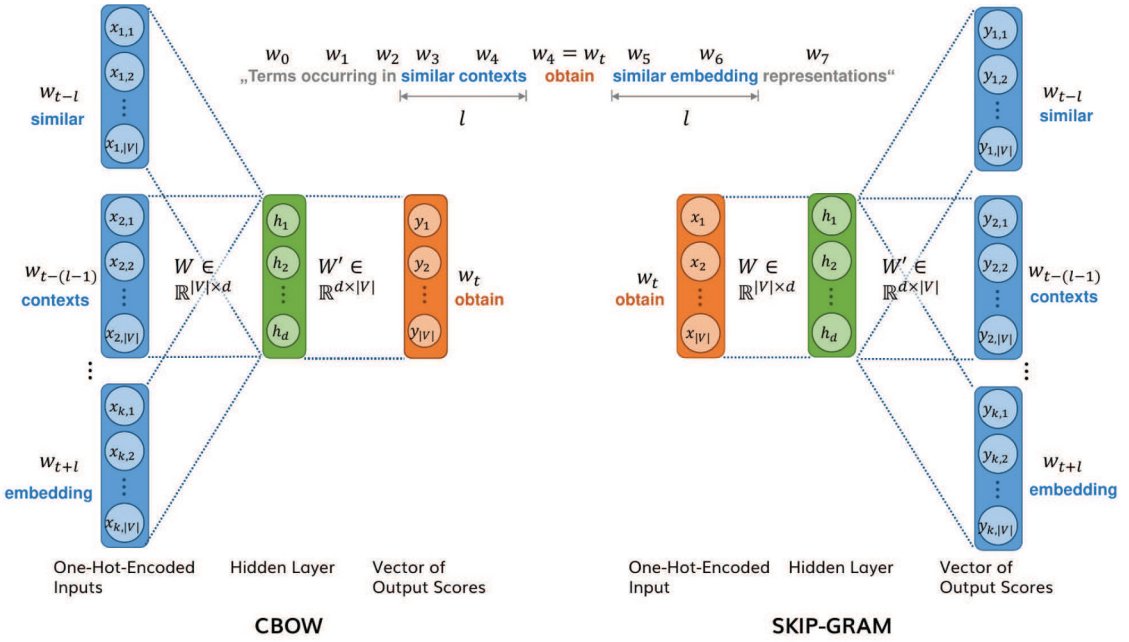


Figure 2.2: Architecture of Word2Vec Neural Networks

Negative Sampling To train word embeddings more efficiently, a different objective is used for training. Thereby, also pairs $(w, C) \in D$ of a center word and a context are considered. The model should then learn to distinguish correct pairs in D from randomly sampled incorrect pairs $(w, C') \in \bar{D}$. The size of $\bar{D}$ is selected to be n times higher than the size of D, where n constitutes a hyperparameter of the model. The following loss function is used:

$$\mathcal{L} = \sum_{(w,C)\in D} \log P(D = 1|w, C) \; + \sum_{(w,C')\in \bar{D}} \log P(D = 0|(w, C') \in \bar{D}) \tag{2.2}$$

For the CBOW model $P(D = 1|w, C)$ is defined as follows:

$$P(D = 1|w, C) = \frac{1}{1 + exp(-\mathbf{w}^T\mathbf{h})} \quad \mathbf{w} \in W' \quad \mathbf{h} = \frac{1}{2l}(\mathbf{x_1} + \ldots + \mathbf{x_k}) \tag{2.3}$$

Hereby, $\mathbf{w}$ is the vector in W' representing the center word w. For the Skip-Gram version, the context words are considered independently from each other. Thus, context pairs with only one context word are considered. The probability $P(D = 1|w, C)$ is derived as follows:

$$P(D = 1|w, C) = \frac{1}{1 + exp(-\mathbf{w}^T\mathbf{h})} \quad \mathbf{w} \in W \quad \mathbf{h} \in W' \tag{2.4}$$

Hereby, $\mathbf{h}$ denotes the vector of the center word. $P(D = 0|(w', c')$ can be defined by its complement $P(D = 0|w', C') = 1 - P(D = 1|w', C')$

Relation to Term-Context Matrix In [LG14b] it is shown that the negative sampling variant of the Skip-Gram model implicitly factorize a term-context matrix $M = [m_{i,j}] \in \mathbb{R}^{|V|\times|C|}$ like it is used by the matrix factorization methods described in Section 2.2.1. This matrix constitutes the product of the term and context embedding matrices defined above: $M = W \cdot W'^T$. Each value $m_{i,j}$ in this matrix corresponds to the product $\mathbf{w_i} \cdot \mathbf{c_j}$ of a term and a context embedding vector. In [LG14b] it is shown that the objective of the Skip-Gram model is minimized when $m_{i,j} = PMI(w_i, c_j) - \log n$ where n is the number of negative samples per word context pair.

GloVe Another popular word embedding model is GloVe [PSM14]. Instead of defining its objective only on the individual context windows, it utilizes global statistics from a co-occurrence matrix. First, it constructs an explicit term-context matrix $X = [x_{i,j}] \in \mathbb{R}^{|V| \times |V|}$ that relates two words w_i and w_j if w_j occurs in the context window of word w_i and holds a global weighted co-occurrence count in $x_{i,j}$. Each center word w_i is associated with a vector $\mathbf{w_i}$ and a bias $\mathbf{b_i}$, and each context word w_j is associated with a context vector $\mathbf{c_j}$ and a bias $\widetilde{\mathbf{b}}_\mathbf{j}$. Then, it samples word context pairs $(w_i, w_j) \in D$, where the matrix X is not zero for training the vectors according to the following loss function:

$$\sum_{(w_i,w_j)\in D} f(x_{i,j})(\mathbf{w_i}^T \mathbf{c_j} + \mathbf{b_i} + \widetilde{\mathbf{b}}_\mathbf{j} - \log x_{i,j})^2 \tag{2.5}$$

Hereby, f denotes a weighting function which assigns co-occurrence counts to values between 0 and 1.

fastText A common problem of the embedding techniques described above is the modeling of rare terms. Embeddings obtained for rare terms are suboptimal because of the little information available to the model. Moreover, for unknown terms, no word embedding can be obtained. To address those problems, [BGJM17] propose an extension of the Skip-Gram model that models terms as a bag of character n-grams. For each word w, a set G_w is constructed that contains the n-grams of w as well as w itself. The authors of [BGJM17] state, it is useful to generate n-grams for all $n \in \{3, 4, 5, 6\}$ in practice. All n-grams of a corpus obtain a vector representation in the first weight matrix W of the Skip-Gram model. The calculation of the probability according to Equation (2.4) is then modified as follows:

$$P(D = 1|w, c) = \frac{1}{1 + exp(-s(w, c))} \quad s(w, c) = \sum_{g \in G_w} \mathbf{z_g}^T \mathbf{v_c} \quad \mathbf{z_g} \in W \quad \mathbf{v_c} \in W' \tag{2.6}$$

Thereby, the dot product $\mathbf{w}^T\mathbf{h}$ of a word vector and a context vector is replaced by the scoring function $s(w, c)$, which represents a word by the n-gram set G_w.

2.2.3 Contextualize Word Embeddings

Two syntactically equivalent words always obtain the same vector by the static word embedding models described above. Thus, a limitation of such models is that they can not cope with polysemy. To solve this problem, contextualized word embedding models take the context of surrounding words into consideration. Early contextualized word embedding models like CoVe [MBXS17] and ELMo [PNI+18] utilize neural networks with an LSTM architecture [HS97] to learn from sequences. Later [DCLT19] propose to use the Transformer architecture [VSP+17] for a contextualized word embedding model called BERT, which leads to large improvements in downstream NLP tasks.

The BERT model is based only on an encoder part of the Transformer neural network architecture [VSP+17]. Originally, the Transformer was proposed for machine translation tasks, where an encoder generates for each input token a vector representation and a decoder predicts a sequence of output tokens base on the input vectors. BERT only uses the architecture of the encoder part. This consists of several so-called *Transfomer blocks* which themselves consist of a multi-head attention layer and a fully connected layer. For more details, we refer to [VSP+17]. The model is pre-trained on a large text corpus and fine-tuned on a task-specific training set. Pre-training requires large computational resources[1]. Fine-tuning is comparably fast and has a lower resource consumption.

[1]The authors stated they run their pre-training procedure for 4 days on 16 TPUs for the largest model.

Input Encoding: Usually, the BERT model gets an input consisting of two sentences. To encode the input text, BERT tokenizes the sentences using a wordpiece tokenizer [SN12, WSC+16] and generates three kinds of input vectors for each token:

1) Token Embedding: Each input starts with the `[CLS]` token, followed by the wordpiece tokenization of both sentences. Between both sentences, a `[SEP]` token is added. For each unique token, a unique embedding representation is generated.
2) Segment Embedding: To encode the information to which sentence a token belongs, segmentation embeddings are added. There are only two different segmentation embeddings for the two different sentences.
3) Positional Embedding: The position of the token is encoded by position embeddings. BERT models have a defined length (typically 500 tokens). Each position has a specific embedding.

The embedding representation of a token is defined by the sum of those three types of embeddings. Before the pre-training, all three embedding types are initialized randomly.

Pre-Training: Pre-training can be done with various pre-training tasks where it is easy to generate a large amount of training data. The most common task is the *Mask-Language Model* task. For this task, an algorithm randomly selects words from sentences in a large text corpus and replaces them with a `[MASK]` token (or a randomly selected word). Then, the model is trained to predict the mask words in each sentence. Hereby, the output embedding at the position of the `[MASK]` token is decoded by an additional fully connected layer with the softmax activation function.

Fine-Tuning: During fine-tuning, the model is trained to solve a specific (NLP) machine learning task. Here a task-specific labeled corpus is required. However, in practice due to the pre-training, a much lower amount of labeled data is required to achieve the same accuracy achieved by a model without pre-training. To fine-tune a model, the BERT-architecture is extended with tasks-specific layers. For simple classification tasks, a layer can be added on the output of the output corresponding to the `[CLS]`. Subsequently, the model is initialized with the weights obtained from the pre-trained model. Afterward, the fine-tuning is done by training the model with the labeled data. The authors test the model on question answering tasks as well as several classification tasks which require language understanding.

2.2.4 Advantages of Contextualize and Static Word Embeddings

In general, contextualized word embedding models deliver superior performance on supervised NLP tasks. On the GLUE (General Language Understanding Evaluation) benchmark [WSM+19], the best-performing models utilize contextualized word embedding models. However, methods for generating static embedding representations are still relevant [GJ21]. For instance, contextualized embedding models might not be effective for unsupervised tasks. To effectively use them, a tasks specific fine-tuning step with training data is required. Embeddings obtained from different layers of a pre-trained BERT model without fine-tuning deliver only poor performance. This is shown in [RG19] for a sentence similarity search task. Here, word embeddings obtained from static embedding models like GloVe [PSM14] produce vector representations superior to embeddings produced by BERT. However, BERT can be fine-tuned to generate static sentence embeddings by providing additional training data, as shown in [RG19]. Further,

in [BDC20, GTZ+20, GJ21], methods are introduced to utilize BERT to generate state-of-the-art static word embeddings. Besides, [PWS19] propose a method to improve a BERT model with static embeddings for question answering. Since contextualized embedding models require a sequence of words as input similar to a sentence, they are less useful for tasks where representations for single words are required. Good examples are text values in tables which usually do not contain whole sentences. Moreover, it is usually hard to effectively combine pre-trained BERT models with other ML models [Edw21]. Thus, for complex ML models, it might be more effective to use static embedding vectors or similarity scores derived from word embedding representations as features. Further limitations derive from the fact that post-processing methods developed for static embedding methods are not applicable for contextualized word embedding models. Examples are the debiasing of word embeddings according to stereotypical semantic relations like gender biases [BCZ+16, KB19] and the so-called retrofitting technique [FDJ+15], which incorporates additional relational information into a word embedding model (see Section 5.1.2). In this thesis, we primarily focus on methods for static word embedding representations. However, this does not limit its applicability to static word embedding models since those representations can be generated with contextualized embedding representations, e.g., with the algorithms proposed in [BDC20, RG19].

2.2.5 Properties of Static Word Embeddings

Word vectors obtained from static word embedding models represent semantic relations between words in the form of arithmetic relations. This can be utilized to quantify similarity and to solve analogy questions [PSM14].

Similarity Quantification

To quantify the similarity between two words (w_i, w_j), one can utilize the cosine similarity between their word vectors $\mathbf{v_i}$ and $\mathbf{v_j}$:

$$sim_{cos}(w_i, w_j) = \frac{\mathbf{v_i} \cdot \mathbf{v_j}}{||\mathbf{v_i}|| \cdot ||\mathbf{v_j}||} \tag{2.7}$$

To speed up the computation for many word pairs, one can normalize all word vectors. This reduces the effort to compute the scalar product of the vectors. To quantify the similarity of word phrases, one can calculate for each phrase the centroid of the word vectors obtained from the tokens of the phrase [ALM17]. Calculating weighted centroids to prefer infrequent words can further improve phrase embeddings obtained in this way.

Word Analogies

In [MCCD13], the authors show that word embeddings can be used to solve so-called analogy tasks. Thereby, semantic relationships such as gender-inflections, and geographical relationships can be recovered [LG14a]. Such an analogy task is formed by two pairs of terms (a, b) and (a', b') sharing a relation. Thereby (a, b, a') correspond to the question and b' to its answer. In an often referred example [MCCD13], ($'man'$, $'woman'$) and ($'king'$, $*$) are given whereby the right answer is $'queen'$. To answer those questions arithmetic operations are applied to the embedding vectors of the given words. Often, the embedding vectors are normalized before performing those arithmetic operations, e.g., in the experiments in [MCCD13]. In [LG14a], the authors describe three different methods to solve analogy tasks: PAIRDIRECTION, 3COSADD, and 3COSMUL.

PairDirection The PAIRDIRECTION method [MYZ13] compares the difference vector of the given word pair (a, b) with all potential word pairs (a', b'). The pair with the highest cosine similarity is selected.

$$b' = \underset{b' \in V \setminus \{a,b,a\}}{\arg\max} \; (sim_{cos}(\mathbf{v_{b'}} - \mathbf{v_{a'}}, \mathbf{v_b} - \mathbf{v_a})) \tag{2.8}$$

In [LG14a], it is stated that the PAIRDIRECTION method performs well in tasks where a restricted set of semantically similar candidate solutions is provided. In contrast, it performs poorly if the embedding vectors of all terms in the vocabulary V are considered because it only models the direction of the relation vector but neglects the distances between the individual embedding vectors.

3CosAdd The 3COSADD method first proposed in [MCCD13] solves the analogy by employing the formula below. Here, all vectors are normalized.

$$\begin{aligned} b' &= \underset{b' \in V \setminus \{a,b,a\}}{\arg\max} \; (sim_{cos}(\mathbf{v_{b'}}, \mathbf{v_{a'}} - \mathbf{v_a} + \mathbf{v_b})) \\ &= \underset{b' \in V \setminus \{a,b,a\}}{\arg\max} \; sim_{cos}(\mathbf{v_{b'}}, \mathbf{v_{a'}}) + sim_{cos}(\mathbf{v_{b'}}, \mathbf{v_b}) - sim_{cos}(\mathbf{v_{b'}}, \mathbf{v_a}) \end{aligned} \tag{2.9}$$

Here, a word b' is determined, which is similar to a' and b and dissimilar to a [LG14a].

3CosMul In [LG14a], a third method called 3COSMUL was proposed, which is similar to 3COSADD but produced better results on several benchmarks.

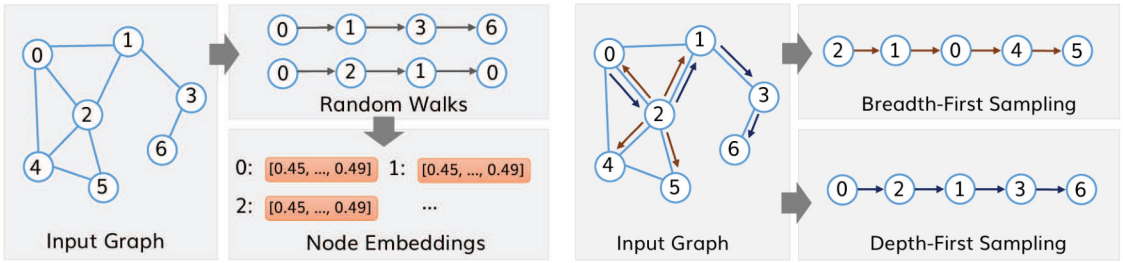
$$b' = \underset{b' \in V \setminus \{a,b,a\}}{\arg\max} \; \frac{sim^*_{cos}(\mathbf{v_{b'}}, \mathbf{v_{a'}}) \cdot sim^*_{cos}(\mathbf{v_{b'}}, \mathbf{v_b})}{sim^*_{cos}(\mathbf{v_{b'}}, \mathbf{v_a}) + \varepsilon} \quad sim^*_{cos}(x, y) = \frac{sim_{cos}(x, y) + 1}{2} \tag{2.10}$$

$$\approx \underset{b' \in V \setminus \{a,b,a\}}{\arg\max} \; \log sim^*_{cos}(\mathbf{v_{b'}}, \mathbf{v_{a'}}) + \log sim^*_{cos}(\mathbf{v_{b'}}, \mathbf{v_b}) - \log sim^*_{cos}(\mathbf{v_{b'}}, \mathbf{v_a})$$

Thereby, the cosine similarity values are mapped to values between 0 and 1, and ε is used to prevent division by zero ($\varepsilon = 0.001$). The authors of the paper argue that the improvement arises from a better balancing of the similarity values by the logarithm.

2.2.6 Node Embeddings

Similarly to word embedding models, a node embedding technique performs a mapping $f : V \to \mathbb{R}^d$ of vertices V in a graph $G = (V, E)$ in a way that geometric relations in $\mathbb{R}^d$ reflect relations of the nodes V in G. The set of node embedding techniques ranges from shallow metric embedding techniques to techniques based on deep neural networks [Gro20]. Traditional metric embedding techniques like Isomap [TDSL00] and Laplacian eigenmaps [BN01] rely on dimensionality reduction techniques. A simple embedding model can be constructed by applying a matrix factorization as described in Section 2.2.1 to the adjacency matrix of the graph. In the resulting set of embeddings, the vectors of two nodes $u, v \in V$ are close if they are directly connected by an edge $(u, v) \in E$. This property is called *first-order proximity* [ZYZZ18].



(a) Deep Walk (b) Node2vec Sampling Strategies

Figure 2.3: Skip-Gram Node Embeddings

Node Embeddings based on Word2vec

In [PARS14], a technique called DeepWalk was proposed to train node embeddings with the Skip-Gram model (see Figure 2.3b). First, this method generates random walks from a graph G. Usually, an equal number of random walks for each node $n \in V$ is generated. Afterward, the random walks are treated like sentences. Using the sliding window approach, pairs (n_c, C) of a center node n_c and a set of context nodes C can be constructed and used to train a Skip-Gram model. The model produces vectors that are similar if the respective nodes occur in similar contexts in random works. In this way, DeepWalk produces embeddings that preserve higher-order proximity [ZYZZ18].

A generalization of the DeepWalk method is the node2vec method [GL16]. The authors identified two kinds of special random walks starting from a node n_0: (1) walks obtained by a *breadth-first sampling* and (2) walks obtained by a *depth-first sampling* (see Figure 2.3b). Node2vec introduces a framework to bias the random walk generation to resemble a breadth-first sampling or a depth-first sampling. Therefore, the authors replace the uniform transition probability used in DeepWalk with a biased transition probability $P(n_x|n_v, n_t)$ for transitions from node n_v to node n_x which depends on the last traversed edge (n_t, n_v) defined in Equation (2.11). Thereby, $d(n_t, n_x)$ refers to the distance between the preceding and the succeeding node, and Z is a normalization constant to fulfill the probability rule of sum. The probability depends on the constants p and q. High values of p and low values of q lead to random walks similar to a breadth-first sampling. In contrast, an opposing assignment of p and q leads to walks similar to a depth-first sampling.

$$P(n_x|n_v, n_t) = \begin{cases} \frac{\pi_{t,x}}{Z} & \text{if } (n_v, n_x) \in E \\ 0 & \text{otherwise} \end{cases} \qquad \pi_{t,x} = \begin{cases} \frac{1}{p} & \text{if } d(n_t, n_x) = 0 \\ 1 & \text{if } d(n_t, n_x) = 1 \\ \frac{1}{q} & \text{if } d(n_t, n_x) = 2 \end{cases} \tag{2.11}$$

DeepWalk is a special case of node2vec with $p = 1$ and $q = 1$. The evaluation in [GL16] shows that different assignments of p and q can lead to significant improvements when the embeddings are used for node labeling and link prediction tasks.

Graph Neural Networks

Besides the Skip-Gram Model, neural networks in form of Graph Neural Networks can be used to train node embeddings, e.g., as done in [KW17]. A definition of a standard GNN architecture is provided in [Gro20]. According to this definition, the GNN holds

for every node V in the graph $G = (V, E)$ a state vector $\mathbf{x_v} \in \mathbb{R}^d$. Those state vectors are modified by the graph neural network using a message-passing algorithm. Therefore, it uses an *aggregation function* α and an *update function* γ. α aggregates the state vectors of the neighbors of a node v into a single vector $\mathbf{a_v}$. Afterward, γ calculates a new node state ${\mathbf{x_v}}^{t+1}$ based on $\mathbf{a_v}$ and the previous state ${\mathbf{x_v}}^{t}$. A simple definition of α and γ is given in Equation (2.12).

$$\alpha(v,t) = \sum_{w \in N(v)} W_{agg} \cdot {\mathbf{x_w}}^{t} \quad \gamma(v) = \sigma\left(W_{up}\begin{pmatrix} {\mathbf{x_v}}^{t} \\ \alpha(v,t) \end{pmatrix}\right) \tag{2.12}$$

Hereby, σ is an activation function. The alternating execution of α and γ is computed for a fixed number of iterations. W_{agg} and W_{up} are weight matrices that are optimized during the training of the GNN according to a specific training objective. More complex GNNs can be built by stacking multiple layers in form of multiple functions α_i and γ_i.

To generate node embeddings, the initial states ${\mathbf{x_v}}^{0}$ are initialized with random vectors and any objective function can be used to train the GNN. The resulting state vectors constitute the desired node embeddings.

2.2.7 Non-Euclidean Embedding Techniques

Usually, word embeddings, as well as node embeddings, are continuous representations in a Euclidean vector space. However, for some applications, other geometric spaces are more suitable.

Hyperbolic Node Embeddings

In [NK17, NK18], an algorithm is proposed to learn node embeddings in hyperbolic vector spaces. The authors observe that Euclidean vector spaces are unsuitable to represent hierarchical data. In order to represent a large highly connected taxonomy graph in a Euclidean vector space such that connected nodes obtain nearby representations and not-connected nodes obtain distant representations, the dimensionality of the vector space needs to be high. In contrast, in a low-dimensional Poincaré ball, this is possible. The authors show that those embeddings are superior in tasks like graph reconstruction and link prediction [NK17].

Binary Word Embeddings

All embedding techniques described above produce continuous representations. Usually, those embeddings are stored with 32-bit floating-point values, e.g., in the popular embedding framework Gensim [RS10]. This leads to relatively large embedding models. Often the embedding vectors require much more memory than the vocabulary, e.g, this is the case in the popular pre-trained Google News Word2vec model[2]. To overcome this problem, some word embedding techniques generate word bit vectors that require less memory and achieve similar quality as traditional word embedding models. Thereby

[2] https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing (Access: 08/26/21)

each word is assigned to a bit vector. In [TGH19], a technique is proposed that compresses pre-trained embedding models with an autoencoder architecture to achieve this. Other models like BinaryBERT [BZH+21] and word2bits [Lam18] directly train binary embeddings. Word2bits uses the word2vec CBOW model with negative sampling and a novel concept called *virtual quantization*. Thereby, the loss function of the CBOW model is modified to apply the quantization function Q (see Equation (2.13)) on each element of center word vector $\mathbf{v}$ and context word vectors $\mathbf{x_1}, \ldots, \mathbf{x_k}$. This function Q assigns values to $-\frac{1}{3}$ or $\frac{1}{3}$.

$$Q(x) = \begin{cases} \frac{1}{3} & \text{if } x \geq 0 \\ -\frac{1}{3} & \text{if } x < 0 \end{cases} \tag{2.13}$$

After the training, the quantization function is applied to the sum of the center word vector and the context word vector for each word to obtain a bit vector. The authors show in [Lam18] that those bit vectors are able to outperform vectors trained with the original CBOW model on several benchmarks. Moreover, in Section 4.5.9, we propose a method to calculate word similarity with word2bit vectors, which is much more efficient than the similarity calculation with traditional word embeddings.

Limitaions of Non-Euclidean Embeddings

Besides the advantages of those non-Euclidean embeddings, a limitation is that tools and frameworks working with embedding models are usually designed for Euclidean embedding models. This limits their utility for real-world applications. For instance, a hyperbolic embedding model can not be used in standard neural network classifiers. Moreover, many index structures for Euclidean vectors are not applicable for hyperbolic and binary embeddings. However, adaptations are possible. For example, multiple neural network architectures for the hyperbolic space have been developed [GBH18].

2.3 EVALUATION OF WORD EMBEDDINGS

Techniques for evaluating word embedding can be categorized into two groups: *intrinsic* and *extrinsic* [SLMJ15]: Extrinsic methods focus on specific applications of word embedding models, such as Named Entity Recognition, Text Classification, Part-of-Speech Tagging, Sentiment Analysis, and many more. Hence, the quality of a word embedding is directly derived from a performance metric applied to the outcome of a downstream task using this embedding. However, [SLMJ15] showed that there is no correlation between word embedding performance for different downstream tasks. For this reason, extrinsic methods cannot be used as an objective metric of word embeddings quality. In contrast, intrinsic techniques investigate properties of word embedding models like the similarity of word vectors and compare those against manually created datasets either constructed in the laboratory or on crowd-sourcing platforms.

Intrinsic evaluation methods can be further distinguished according to the properties they focus on. Often considered are the correlation of vector *similarity* with similarity observed by humans, the accuracy observed by deriving *analogies* from arithmetic properties of word vectors, and *cluster-based* properties. An overview of the most commonly used data collections is given in Table 2.1.

Dataset	Size	Comment
MEN [BBBT12]	3,000 word pairs	general domain knowledge, semantic relatedness, scale from 0 to 50
SimLex-999 [HRK15]	999 terms	general domain knowledge, semantic similarity, scale from 1 to 10
MTurk-771 [HDGK12]	771 word pairs	semantic relatedness, scale from 0 to 5
RG-65 [RG65]	65 word pairs	semantic similarity, scale from 0 to 4
RW [LSM13]	2,034 word pairs	semantic similarity of *rare words*, a scale from 0 to 10
SimVerb-3500 [GVH+16]	3,500 word pairs	only verbs, semantic similarity, scale from 0 to 4
TR-9856 [LDH+15]	9,856 word pairs	focusses on relatedness of multi-words
WordSim-353 [FGM+01]	353 word pairs	focus on relatedness, scale from 0 to 10

(a) Similarity Datasets

Dataset	Size	Comment
WordRep [GBL14]	≈118M analogies in 26 semantic classes	Google Analogy + data from WordNet
Google Analogy [MCCD13]	19,544 analogy tasks build from 550 relations, 14 categories	semantic and syntactic relations, mostly city-country relations
MSR [MYZ13]	8,000 analogies in 16 morphological classes	syntactic relations

(b) Analogy Datasets

Dataset	Size	Comment
ESSLLI-2008 [Ass08]	45 terms, 9 categories	every category encompasses 5 verbs
8-8-8 [CCN16]	128 terms, 8 categories	constructed for outlier detection
WikiSem500 [Gam19]	500 cluster of 9 - 14 terms per language	outlier detection tasks constructed from Wikidata (multi-lingual)

(c) Cluster-based Datasets

Table 2.1: Data Collections for Intrinsic Evaluation

2.3.1 Similarity Evaluation

Word embeddings are often evaluated by similarity metrics. Datasets like MEN [BBBT12] or SimLex-999 [HRK15] contain similarity ratings of human judges. Based on those ratings, one can construct a ranking of similar words to an initially selected word. Accordingly, a ranking can be constructed by the word embedding model based on the cosine similarity of the word representations (see Section 2.2.5). Then, the ranking correlation coefficient of both rankings serves as a quality score of the word embedding model. However, similarity-based quality assessments heavily depend on the notion of similarity, which is different for all evaluation datasets. For example, in SimLex-999 semantic similarity was in the focus of consideration, while MEN also assigns higher scores to related words. Even though similarity-based evaluation is very popular, it is often criticized because similarity is very subjective and biased by certain factors [Bak18].

2.3.2 Analogy Evaluation

The analogy task evaluates if certain relations can be recovered by a given word embedding model. Thereby, analogy queries given by a gold standard are solved by a word embedding model. Hereby, one of the three methods described in Section 2.2.5 can be used. For each question, the result is compared to the solution provided by the gold standard. After all analogy questions are solved, an accuracy value is derived which can be compared to accuracy values of other embedding models on the same dataset.

There is a variety of publicly available datasets with analogy queries for word embedding models. However, a limitation is that those datasets do not allow a domain-specific evaluation. The largest data collection is WordRep [GBL14] containing over 118M analogies derived from WordNet. The analogies are grouped according to WordNet relations such as Antonym, MemberOf, IsA, etc. While WordRep is a very valuable data collection to test word embeddings, we noticed that the abstraction level of the relationships is too high for detailed domain-specific evaluations. Other common analogy datasets are Google Analogy and MSR: Google Analogy [MCCD13] is a subset of WordRep where over 50% of the analogies are encompassed by country-city relation pairs. MSR [MYZ13] focuses on syntactic relations.

2.3.3 Cluster-based Evaluation

In [BDK14], a clustering approach is applied to the vectors of a word embedding model to categorize the respective words in groups. The resulting clusters are compared with groups given by a gold standard using a set similarity metrics. However, this approach is criticized because it could depend on the clustering algorithm and its initialization [Bak18]. As an alternative, [CCN16] propose to evaluate word embeddings on the outlier detection tasks and publish the 8-8-8 dataset for this purpose. Here, the goal is to determine the outlier for a given set of terms. Using the word embedding model the term with the highest vector distance to the centroid of word vectors of the provided terms is selected. This should coincide with the outlier in the gold standard. This method evaluates the cluster property of a word embedding model independently from a specific clustering algorithm.

2.4 APPLICATION FOR TABULAR DATA

Word embeddings have already been applied to text values in tabular data for various data management applications. At this point, we want to give a brief overview of the main application areas.

2.4.1 Semantic Search

Word embeddings can be used to enhance the capabilities of search applications. Traditionally, many of those applications rely on the popular vector space model [SWY75]. This requires syntactic similarity of words, i.e., search results should contain the words in the query or syntactically similar words [ZD95]. A major problem with those techniques is that semantically similar or related words are not necessary syntactically similar. To handle this problem, handcrafted thesauri supply mappings between synonyms [Voo94]. However, such thesauri provide only limited coverage of domain-specific terms. In contrast, word embedding models provide a data-driven solution to obtain high-quality similarity scores. Therefore, they can be employed to quantify similarity [ALM17] and semantic distance [KSKW15] of text to implement semantic search features. This can also be utilized for text in tabular data. Thus, word embeddings have been used to implement semantic SQL queries [BS17] and search systems for unionable tables [NZPM18]. Moreover, in [HPW21, HPR+21], word embeddings are used to answer queries with quantity filter predicates over semi-structured content. Here, quantity facts extracted from Web tables are matched to query terms based on word embedding vector distances. Besides, using embedding models pre-trained on text [HNM+20, YNYR20] pre-train contextualized embeddings with a Transformer architecture jointly on text and tabular data for semantic parsing tasks on tables like question answering.

2.4.2 Data Curation

Text values in natural language contain references to real-world entities, which are independent of the dataset in which the values occur. In contrast, ids are often specific for a particular dataset, and numbers need to be connected to a unit and an attribute to be interpretable. Therefore, text values are specifically valuable for data integration and curation tasks, e.g., the linking of entities across datasets [KDSG+16] and missing value imputation [BSS+18]. Pre-trained word embeddings capture knowledge derived from the large corpora they are trained on. Several entity matching systems exploit this knowledge [ETJ+18, MLR+18, BS20, LLS+20a, PB21], leading to substantial improvements when compared to traditional approaches based on heuristics especially for matching textual and dirty data records. Recently, the authors in [TFL+21] propose pre-training embedding models on tuples of tables to employ them for several data cleaning tasks.

2.4.3 Data Discovery

Knowledge from pre-trained word embeddings can also be utilized for data discovery. When tokens of two text values are transformed into sets of word vectors, those word vectors exhibit a pair-wise high cosine similarity. This has been exploited by [FMQ+18] to find links between tables across datasets. Similarly, in [NPZ+20], the cosine similarity of mean embedding vectors of text values in columns is calculated. Subsequently, this

similarity is used as a metric in a probabilistic model to obtain a suitable hierarchical navigation structure for organizing tables in data lakes. The resulting navigation structure supports users in discovering topic-related tables in the data lake. In [DTXO21], the authors propose a framework to find joinable tables in data lakes. Therefore, the framework transforms text values in the tables into vectors by using a word embedding model and index them to efficiently search. Afterward, columns with a high amount of text values with high similarity according to the embedding vectors are considered joinable.

Further, many data discovery systems use word embedding models to encode text as input for ML models to annotate data. For instance, in [YPS+20], a data discovery system is proposed to extract PDF tables and assemble their content in a large master table in a database. Here, the system uses the word embedding representations of extracted text values to assign them to concepts corresponding to columns in the master table. TCN [WSL+21] is a system for column type prediction and prediction of relation types between columns to extract knowledge. Here, word embedding models are used to encode text in cells. To process spreadsheets, [GGPS20] annotate cell types with a model initialized with word embeddings. Besides directly encoding inputs with word embeddings, [DSL+20] proposes a Transformer model with parts of the model initialized with weights of a contextualized word embedding model. Afterward, this model is pre-train on tabular data for table interpretation tasks.

3

SYSTEM OVERVIEW

In this chapter, we provide an overview of the our research objectives on integrating word embeddings in relational database systems. This involves the implementation of *(a) database queries involving embedding representations* and infrastructure for *(b) the storage and efficient retrieval of word embedding representations*. Moreover, we investigate techniques to optimize embedding representations of text values in database systems by *(d) a novel context adaptation algorithm*. Besides, we provide *(d) support for selecting a word embedding model* suitable for a user's application. Furthermore, we propose *(e) novel embedding techniques to be pre-trained on tabular data*. Those techniques model semantic relations arising from the alignment of words in tabular layouts that can only hardly be derived from text documents. Thus, many applications working with tabular data profit from using those embeddings instead of word embeddings trained on text.

We begin with motivating the integration of word embeddings in database systems in Section 3.1, investigate the characteristics of common pre-trained word embedding models in Section 3.2, and examine the main objectives of such an integration in Section 3.3. Afterward, we take a specific look at word embedding operations in Section 3.4. In the following, we derive detailed requirements to design a system satisfying the objectives introduced before. Therefore, we investigate performance optimizations in Section 3.5, context adaptation in Section 3.6, and model recommendation in Section 3.7. Finally, we look at opportunities and requirements of training embeddings on tabular data in Section 3.8

3.1 OPPORTUNITIES OF AN INTEGRATION

Pre-trained word embeddings constitute a rich source of extended common knowledge in the domain of the text they are originally trained on. Applications can utilize this valuable knowledge either by employing embeddings in supervised machine learning models or through the application of arithmetic operations (see Section 2.4). To support applications to benefit from word embeddings, we aim at extending the capabilities of traditional relational database systems which are still by far the most common DBMSs. On the one hand, the database system should support machine applications by providing an interface to efficiently access embeddings. On the other hand, the database should be extended by novel text operations based on arithmetic word embedding operations to enable applications to exploit the knowledge encoded in word embeddings for semantic queries. Further, applications can be supported by recommending embedding models and algorithms that optimize embedding representations of text values to better model their context-specific meaning in the database.

```
SELECT keyword
FROM keywords,
ORDER BY cosine_similarity(
 'comedy',keyword) DESC
```

The similarity of keywords to the term "comedy" is quantified, and the keywords are sorted accordingly. The most similar keywords are "sitcom" and "sketch_comedy".

(a) Quantify Similarity

```
SELECT kNN(p.name, 5)
FROM directors AS d
INNER JOIN persons AS p
ON p.id = d.director_id
```

The query retrieves the five most similar terms to the directors in the database. For "Quentin_Tarantino", this results in other Hollywood directors like "Martin_Scorsese", and movies directed by Tarantino, e.g., "Pulp_Fiction".

(b) Most Similar Operation

```
SELECT m.title, t.term, t.score
FROM movies AS m,
 kNN(m.title, 3, (
 SELECT title FROM movies)) AS t
ORDER BY m.title ASC, t.score DESC
```

The query determines the three most similar movies to each movie title. In the case of the movie "Godfather", this leads to "Scarface", "Untouchables", and "Goodfellas".

(c) Resticted Most Similar Operation

```
SELECT g.term, g.group_term
FROM groups(
 (SELECT title FROM movies),
 'comedy,horror,documentary')AS g
```

The query groups movies to the categories "Documentary", "Comedy", and "Horror". Here, "Toy_Story" and "Big_Lebowski" get grouped into "Comedy" while "Planet_Earth" gets assigned to "Documentary" and "Psycho" to "Horror".

(d) Group Text Values

```
SELECT term, group_id
FROM cluster(
 SELECT keyword FROM keywords)
ORDER BY group_id ASC, term DESC
```

The query assigns keywords into groups of similar terms. For instance, "robbery" and "murder" are assigned to one, and "millionaire" and "chauffeur" to another group.

(e) Cluster Operation

```
SELECT analogy('Godfather',
 'Francis_Ford_Coppola',m.title)
FROM movies AS m
```

The query provides the movie title "Godfather" and its director "Francis_Ford_Coppola" to the analogy operation. Then, this relationship is investigated for other movie titles. For the movie title "Inception", this query returns the name of its director: "Christopher_Nolan".

(f) Analogy Operation

```
SELECT c.name,
 analogy('USA', 'English',
 name, (SELECT * FROM languages))
FROM countries AS c
```

The query assigns country names to languages. For the "Netherlands", this query returns "Dutch".

(g) Resticted Analogy Operation

```
SELECT j.term1, j.term2
FROM kNN_join(
 (SELECT title FROM movies
 WHERE release_data = 2010), 3,
 (SELECT title FROM movies))AS j
```

As in Figure 3.1c, similar movies are obtained for titles in the movie table. However, this is done only for movies released in 2010. The `knn_join` operation is used instead of the `knn_in` operation to more efficiently execute the query. For "Shutter_Island" the query returns "Inception", "Batman_Begins", and "Disturbia".

(h) kNN Join Operation

Figure 3.1: Examples of Word Embedding SQL Queries (simplified)

Word Embedding Operations: Examples of arithmetic word embedding operations have been already described in Section 2.2.5. In the context of the schema of a movie database[1] with real-world data from TMDB, this allows a user, for instance, to perform similarity queries on movie titles. Moreover, the user can utilize the information encoded in the word embeddings to reveal semantic relations between the text values in

[1] https://github.com/guenthermi/the-movie-database-import (Acess 03/22/21)

the database. An overview of novel query types implemented with word embedding operations is shown in Figure 3.1c. The capabilities of word embedding operations go far beyond string functions provided by traditional database systems. For instance, word embeddings enable domain-specific and multi-lingual similarity queries, data-driven solutions for query expansion, and several other novel query capabilities. Section 3.4 gives a detailed overview of the definitions of all operations and compares embedding operations to traditional text processing operations implemented in database systems.

Fast Access Methods for Embeddings: Word Embedding models are typically relatively large sets of vectors. Storing a model requires several gigabytes of storage. Therefore, storing the whole model in memory might be impractical. Here the database can assist an application with efficient access methods for word embeddings stored on disk. To obtain an embedding representation for a text value not in the vocabulary with embedding models based on word2vec and GloVe, it is necessary to calculate vector representations with the averaging method [ALM17]. The DBMS can implement operations to calculate those vectors and manage them in the database. Moreover, to efficiently execute the operations described above, it is necessary to implement special index structures and search algorithms. Therefore, integrating these indexes and algorithms into the database can boost the performance of word embedding applications.

Optimization of Embeddings: Applications working with text in tabular data frequently utilize word embeddings to represent text. However, word embedding techniques are usually designed for text documents and are not optimized for text in relational database systems. With this in mind, we identified opportunities to optimize the application of word embeddings on text in RDBMSs. In particular, we aim at incorporating the relational context of text values in the database in the vector representation of a textual value. Applications using embeddings for text in database systems can provide from this optimization process. Specifically, ML models achieve higher accuracy by using such optimized embeddings.

Model Recommendation: Different embedding models capture different notions of similarity. Based on the application focus, one should decide on an embedding model. However, text corpora used for training an embedding model, e.g., the complete English Wikipedia, are usually too large to be overseen by a human user. Routines for recommending models can assist the user. In particular, we aim at designing a data-driven solution to generate a benchmark. Based on this benchmark, the user should be able to identify the domains of text values represented in a model.

Tabular Embeddings: While the application of word embedding models on tabular data has been extensively motivated above, not all semantic relations between words in tables are captured by traditional word embedding models. In particular, categorical relations typically existing between text in the schema of a database table and the actual instance data are in general only poorly represented. Moreover, text values in tables do not occur in a sentential contexts but rather in a tabular context. This is also reflected in different methods to format text in cells that are unknown to word embedding models. By designing novel algorithms to train word embeddings on tabular data, we aim to overcome those limitations and improve the performance of applications using word embedding models to model text in tables.

	word2vec Google News[1]	GloVe Common Crawl[2]	fastText Wiki[3]	word2bits Wiki[4]
Abbr.	W2V-GN	GV-CC	fT-W	W2B-W
Size	3,000,000	2,196,017	–	400,000
Dim.	300	300	300	800

[1] https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing (Access: 03/31/21)
[2] http://nlp.stanford.edu/data/glove.840B.300d.zip (Access: 03/31/21)
[3] https://dl.fbaipublicfiles.com/fasttext/vectors-wiki/wiki.en.zip (Access: 03/31/21)
[4] https://drive.google.com/open?id=107guTTy93J-y7UCO2ZA2spxRIFpoqhjh (Access: 03/31/21)

Table 3.1: Model Characteristics

Existing Systems for Managing Word Embeddings: There are several frameworks to load word embedding models and execute word embedding operations on text values. Examples are Gensim [RS10] and the fastText library[2] for models trained with the fastText embedding technique [BGJM17]. Moreover, there are implementations of Web services, e.g., word2vec-api[3] and bert-as-service[4], to encode text into embedding vectors and execute embedding operations. As far as we know, there is only one related research work that integrates word embedding operations in a database [BS17, BBS17] which is called a *cognitive database* by the authors. Besides, [TYS+21] makes use of the strong performance of contextualized word embeddings in question answering to develop a new type of database system called *NeuralDB* for text in natural language.

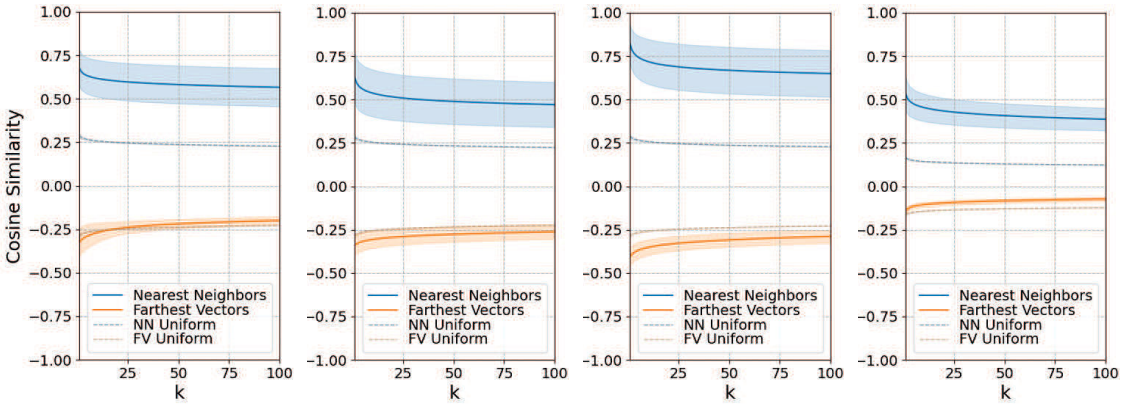
Cognitive Database: In [BS17] the authors integrate word embedding operations in Apache Spark[5]. This system provides additional text operations in the form of user-defined functions (UDFs) to calculate the similarity of text values based on the cosine similarity of the respective word vectors and solve analogy questions. All UDFs are implemented in Python. Those UDFs extend the abilities of SparkSQL. The authors call SQL queries using those word embedding operations *cognitive intelligence queries* to highlight the novel types of queries enabled by word embedding operations. To execute the operations, the UDFs access the embeddings stored in a separate system table. The embedding tables are either created from a pre-trained word embedding model, or an embedding model is trained directly on the database. To directly train embeddings, the authors propose a technique to serialize text sequences from the tables in the database, which we describe in more detail in Section 7.1.1.

NeuralDB: Contextualized word embedding models like BERT [DCLT19] pushed the state-of-the art in question answering tasks significantly. In [TYS+21] the authors utilize this ability to build a database system, which can answer questions in natural languages given by a user based on documents stored in the database system. It stores text snippets and encodes them as embedding vectors. The system determines embeddings similar to the embedding of the question to find relevant snippets to predict an answer using a machine learning model. While this is an interesting application of word embedding models, it does not support systems that use word embeddings beyond question answering.

3.2 CHARACTERISTICS OF WORD VECTORS

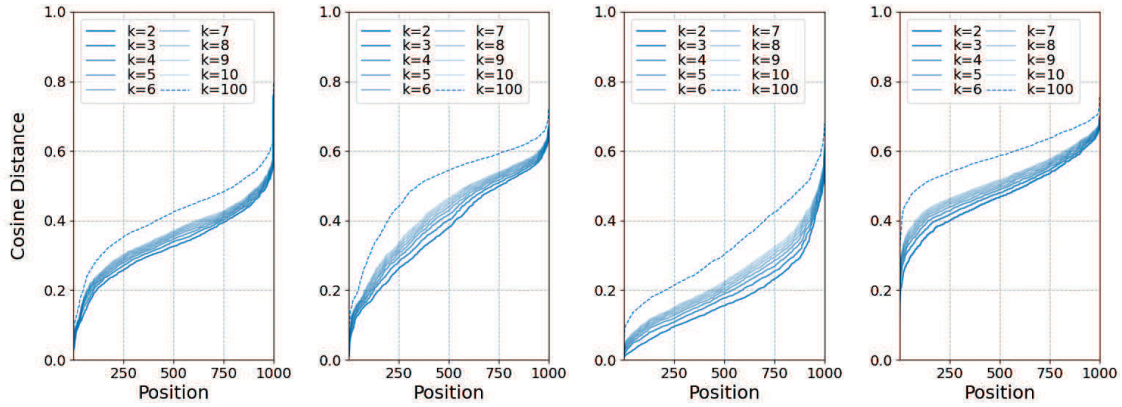
To identify the requirements of handling word embeddings in database systems, it is important to understand the characteristics of word embedding representations. High-

[2] https://fasttext.cc/ (Access: 03/23/21)
[3] https://github.com/3Top/word2vec-api (Access: 03/23/21)
[4] https://github.com/hanxiao/bert-as-service (Access: 03/23/21)
[5] http://spark.apache.org/ (Access: 03/24/21)



(a) word2vec Vectors (b) GloVe Vectors (c) fastText Vectors (d) word2bits Vectors

Figure 3.2: Cosine Similarity to Nearest Neighbors and Farthest Vectors



(a) word2vec Vectors (b) GloVe Vectors (c) fastText Vectors (d) word2bits Vectors

Figure 3.3: k Distance Diagrams

dimensional data tend to be sparse and uniformly distributed due to the curse of dimensionality [Bel57]. Accordingly, the data points tend to be pair-wise very different and it is hardly possible to organizing similar data points into groups. This effect does already occur in vector spaces with only about 15 dimensions [BGRS99]. The authors of [BGRS99] further show that linear scans often outperform index structures for multi-dimensional data with more than 10 dimensions due to this phenomenon.

To examine if this effect also occurs in word embedding datasets, we investigated several popular pre-trained word embedding models with 300 - 800 dimensions listed in Table 3.1. All embedding models represent words with vectors of much more then 15 dimensions. We sampled $1,000$ vectors of each word embedding model, and determine for each vector the $k = 100$ closest vectors (nearest neighbors) as well as the 100 farthest word vectors according to the cosine distance. Since fastText generates vectors based on their subwords, there is no static set of vectors. Thus, we generate the embeddings for all terms in the vocabulary of the word2vec model with fastText to obtain a representative vector set. Figure 3.2 displays the distributions of the cosine similarity values of the nearest and farthest vectors[6]. It compares this to the distributions of sets of uniformly sampled vectors with the same size and dimensionality. As one can see, the difference between

[6]The solid line indicates the mean and the transparent area the standard derivation.

the similarity of the nearest neighbors and the similarity of the farthest vectors is significantly greater for all word embeddings in comparison to the uniformly sampled vectors. This indicates that word vectors tend to build clusters to some extend. Nevertheless, the cosine distance of a vector to its nearest neighbors is still relatively high and increases only marginal with a higher value of k. This indicates that indexing word vectors for exact search is still challenging because there are no dense clusters. The word2bits dataset has the highest dimensionality. Accordingly, the difference between nearest neighbors and farthest neighbors is the lowest. This also signifies that word embedding models are affected by the curse of dimensionality.

We further investigate the average cosine distance of the vectors to their nearest neighbors by creating *k distance diagrams* displayed in Figure 3.3. Therefore, we determine the average distance for each vector for several different values of k and sort the distance values. One can see that the distances continuously slightly increasing with the position in the list while most of the values concentrate around a median value. For the DBSCAN algorithm [EKS+96], the k distance diagram is used to determine a threshold ε to separates noisy points with an average distance above ε from points in clusters. The threshold ε is determined by detecting a point of a sudden increase of the distance values in the graph. In all the graphs in Figure 3.3, the distances increase smoothly, indicating that there is no clear separation between vectors that are part of dense clusters and vectors outlying the clusters.

In summary, we can derive the following rationals from this evaluation:

- The curse of dimensionality does apply to word vectors. Thus, indexing techniques for exact retrieval are not promising and methods utilizing approximation should be considered.
- In comparison with uniformly distributed data, word vectors tend to form clusters. Thus, distribution-aware index structures should be used.
- In the k distance diagrams, we can not recognize a separation of noisy data points. Moreover, the average distance values concentrate around a median value.

3.3 OBJECTIVES AND CHALLENGES

Figure 3.4 provides an overview of the desired extensions of a database system. Those implement the five objectives mentioned at the beginning of the chapter:

A **Word Embedding Operations:** The system should provide word embedding operations that can be combined with the SQL query language similarly as in the examples in Figure 3.1. Essential operations are similarity quantification, analogy calculation, grouping, and similarity joins. We identified that all those types can be derived from variants of the kNN-Join [YLK10] operation. Thus, other operations can be implemented by driver functions using a kNN-Join operation.

B **Efficient Storage Engine:** The system should be able to store word embedding representations of text values next to the actual structured data. Word embedding models can consist of hundreds of millions of floating-point values. Thus, the system should store those models on disk and provide fast access to single embedding vectors.

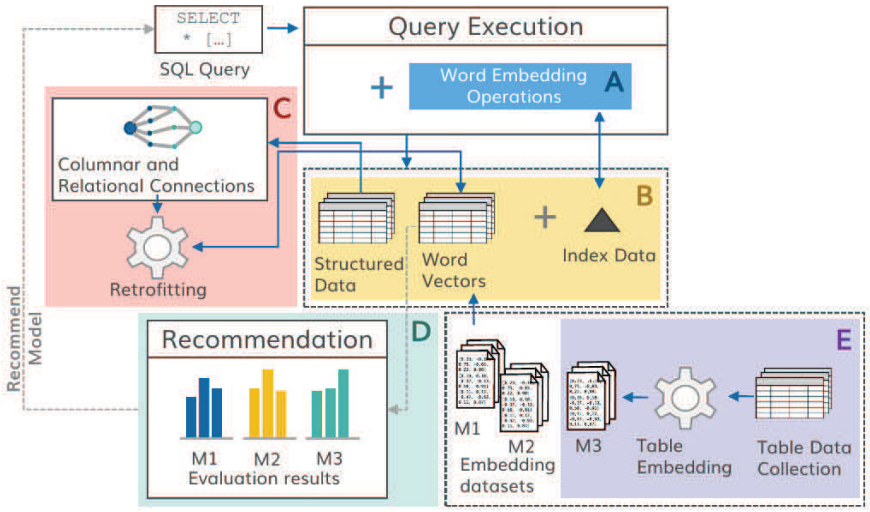


Figure 3.4: Management of Embedding Representation in an RDBMS

To efficiently execute operations, a system should provide index structures suitable for searching word embedding vectors. Since such embedding vectors are high-dimensional, index structures for exact search like k-d trees [FBF77] are not appropriate [SAH08] (see Section 3.2). However, since state-of-the-art word embedding methods generate learned representations that do not constitute an ideal set of word vectors, it is appropriate to use index structures for approximated nearest neighbor (ANN) search.

C **Embedding Optimization** A naïve application of a word embedding model is not sufficient to represent the meaning of text values in a database which is often more specific than the general semantic encoded in the raw word embedding. This leads to sub-optimal embedding representations of text values a potentially undesired results of word embedding operations when applied to the text values in the database. Thus, an algorithm should utilize the information given by the disposition of the text values in the database schema, e.g., which words appear in the same column or are related, to improve the embedding representation.

D **Adaptivity:** The system should provide methods to switch between word embedding models underlying the execution of the embedding operations. Different word embedding models are suitable for text values of different domains. To select a suitable model, the system should recommend models to the user based on a domain-specific evaluation of the embedding models.

E **Pre-Trained Table Embeddings** Word embedding models are trained on sequences of words (i.e., sentences), while text in tables is arranged in a grid format and appears in the form of schema and instance data. The common word embedding model can not distinguish between a text value appearing in the table body and the same text value in the column title and thus, assign the same embedding representations to both values. Moreover, semantic relations obtained from the alignment of text values in tables are often poorly represented in word embedding vectors. Since many applications working with tabular data and using word embeddings can profit from such relations and separate embeddings for schema and instance data, we intend to design an embedding technique for tables compatible with our system.

3.4 WORD EMBEDDING OPERATIONS

Based on the properties of static word embeddings (see Section 2.2.5), it is possible to implement several novel operations for text values in database systems. Our goal is to integrate those operations into a relational database systems. Thereby, we largely extend the capabilities offered by traditional SQL query interfaces.

Definition of Operations The primary word embedding operations are the calculation of similarity values and the answering of analogy questions. Based on this functionality, we implement the following word embedding operations where different interfaces are provided for each function[7]:

- *cosine_similarity(value_1 varchar, value_2 varchar)*: This function quantifies the similarity between two text values (see Figure 3.1a).
- *kNN(input varchar, k int)*: It searches for the k most similar text values according to the input value (see Figure 3.1b).
- *kNN_in(value varchar, k int, output_set varchar[])*: It searches for the k most similar text values to the input value in a restricting set of output text values, e.g., to obtain only results in a specific column in a database relation (see Figure 3.1c).
- *groups(values varchar[], groups varchar[])*: The input text values are assigned to groups specified in the second set of text values according to their similarity (see Figure 3.1d).
- *cluster(values varchar[], k int)*: Input text values are clustered into groups by applying the k-means algorithm on their embedding representations (see Figure 3.1e).
- *analogy(value_1 varchar, value_2 varchar, value_3 varchar)*: It solves analogy queries using the *PairDirection*, *3CosADD*, or *3CosMul* method [LG14a] (see Figure 3.1f).
- *analogy_in(value_1 varchar, value_2 varchar, value_3 varchar, output_set varchar[])*: This operation solves Analogy queries, where the result set is restricted to a specific set of output text values (see Figure 3.1g).
- *knn_join(query_set varchar[], k integer, target_set varchar[])*: This function performs for each text value in the first argument a kNN search in the target set (see Figure 3.1h).

Capabilities of Word Embedding SQL Queries The SQL query language itself provides limited capabilities to compare text values. On the one hand, one can apply the equal and the `LIKE` operator to text values to check syntactic equivalences. On the other hand, popular RDBMSs like PostgreSQL provide string functions for comparing text values, e.g., `regexp_match` allows the user to identify substrings matching a regular expression.

Besides, search features have been implemented and integrated into database systems. For example, for PostgreSQL, a Full-Text-Search extension[8] exists. Moreover, ontologies are used to enable new query types in RDMBSs [DCES04, LWW13], e.g., to identify rows with text values that are semantically related in the ontology.

[7] The results stated for the examples are obtained from the W2V-GN model in Table 3.1

[8] `https://www.postgresql.org/docs/13/textsearch.html` (Access: 04/21/21)

Word embedding operations enable several novel query capabilities going far beyond the functionality provided by such traditional text query features:

Context-Sensitive Representation: Employing different domain-specific word embedding models, enables domain-specific notions of similarity. In this way, context-sensitive similarity queries are possible.

Data-Driven Similarity: While traditional similarity queries rely on human-generated thesauri, word embeddings provide a data-driven solution to quantify similarities of short texts. Since popular pre-trained word embedding models contain large vocabularies, they cover a much larger set of terms than most thesauri provide. Moreover, subword-based models can quantify similarity for almost any text value pair.

Inductive Reasoning: The operations reduce the demand for explicit knowledge required in a database. For example, it is not necessary to provide a mapping between cities and languages in the database. Instead, the mapping can be obtained by an analogy query like the one in Figure 3.1g.

Complex Similarity Queries: Beside simple queries of word similarity and relatedness, word embeddings allow more complex query types, e.g., analogy queries. Especially for data discovery, those operations constitute valuable extensions to the traditional keyword search features provided, for example, by the PostgreSQL full-text search and several data discovery tools [ACD02, BBN19].
To implement semantic search on multi-media data, e.g., videos, images, and audio files, computer vision and speech recognition tools can tag multi-media objects with words and phrases describing those objects. Then, one can apply word embedding operations on those tags. In [BBS17], the authors used the IBM Watson Visual Recognition Service to obtain tags for images. Then, these tags are used to train word embedding models and execute semantic queries.

Multilingualism: It is easy to support multiple languages by adding word embedding models for them. On the fastText website, pre-trained models for over 150 languages are available[9]. Moreover, in [MLS13], the authors show how a translation matrix can be trained to transform embeddings of an embedding model in one language to vectors in an embedding model in another language. This enables cross-lingual similarity search. Further, in [VM15], a method is proposed to train directly bilingual word embeddings.

Requirements for the Integration of Word Embedding Operations For the integration of word embedding operations into a relational database system, we identified the following requirements:

A1 **SQL Compatible:** All operations should be implemented in a way, that the user can use them together with the SQL query language.

A2 **Flexible Scope:** The scope of the operations should not be always be a fixed target set but could be provided as an argument to the operations themselves, e.g., to support `kNN_in` and `analogy_in`.

A3 **Flexible Interfaces:** Different interfaces should be provided for the operations, e.g., to support vector inputs, as well as textual inputs.

[9] `https://fasttext.cc/docs/en/crawl-vectors.html` (Access: 04/27/21)

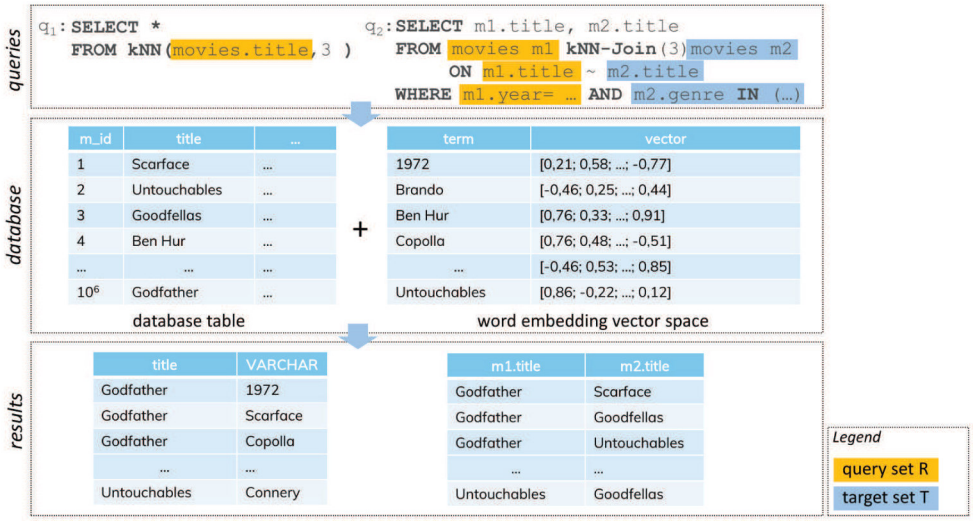


Figure 3.5: Two Example Queries: kNN-Search and kNN-Join

A4 **Model Independent:** The operations should be independent of a specific embedding model. Thus, the system should enable the user to switch the word embedding model underlying the execution of the operations as also required by objective E in Section 3.3.

A5 **Configuration via GUI:** The user should be able to adjust parameters and the configuration of the operations via a graphical user interface, e.g., to change the accuracy when word embedding calculations are approximated.

3.5 PERFORMANCE OPTIMIZATION OF OPERATIONS

As stated in Section 3.3 most word embedding operations can be implemented via kNN-Joins. Thus, to provide efficient word embedding operations, it is especially important to efficiently execute kNN-Join operations. The kNN-Join is a generalization of the K nearest neighbor search (kNN search) operation, a universal data processing technique and also a fundamental operation for word embeddings trained by word2vec or related approaches.

Given a set of query vectors $R \subset \mathbb{R}^d$ and a set of target vector $T \subset \mathbb{R}^d$, a kNN query with query vector $\mathbf{r} \in R$ in the Euclidean vector space is defined as follows:

Definition 3.5.1. *The kNN query of* $\mathbf{r}$ *over* T*, noted* $kNN(\mathbf{r}, T)$*, can be defined as:*
$kNN(\mathbf{r}, T) = \arg\min_{\{\mathbf{t_1},\ldots,\mathbf{t_k}\} \in T^{[k]}} \sum_{i=1}^{k} d(\mathbf{r}, \mathbf{t_i})$.

Here d denotes the distance function between two elements. Typically, in the context of word embeddings, the cosine distance is used. However, if all vectors in R and T are normalized, the cosine distance is proportional to the squared Euclidean distance. The normalization of the vectors does not change the cosine distance. Thus the $kNN(\mathbf{r}, T)$ for any $\mathbf{r}$ and T can be computed using both metrics.

If the query is not just one element but instead a set, the operation is denoted as kNN-Join.

Definition 3.5.2. *The kNN-Join between a query set R and a target set T is defined as:* $kNN(R \ltimes T) = \{\langle \mathbf{r}, \mathbf{t} \rangle | \mathbf{t} \in kNN(\mathbf{r}, T), \mathbf{r} \in R\}$.

Examples for the usage of kNN and the kNN-Join operations in the context of a word embedding database system is shown in Figure 3.5. While the left query uses the kNN operation to determine for each movie three terms with similar embeddings to the embeddings of movie titles, the kNN-Join query restricts the set of query terms to titles of movies released in a specific year and the set target terms to titles of movies of specific genres.

Requirements for kNN-Join in Word Embedding Space We aim at providing a kNN-Join that is particularly suitable for high-dimensional data and varying target sets. In detail, we identify the following requirements to be solved by index structures and the kNN-Join operations:

B1 **Minimization of index accesses:** Since the word embedding data is stored in database relations on disk, accessing data is time-consuming. Thus, the join operation should minimize the number of word vectors and the index data that needs to be retrieved. Often kNN-Joins are implemented by executing multiple kNN queries. In this case, it should be prohibited to access the index separately for each query.

B2 **High-dimensional data:** Previously research on kNN-Joins for relational database systems focuses mostly on low-dimensional data [YLK10]. Because of the *curse of dimensionality* [Bel57], techniques for exact kNN-Joins, trying to hierarchical partition vector spaces, cannot be applied efficiently (see Section 3.2). Fortunately, for our system approximation is appropriate as mentioned in Objective B in Section 3.3. Hence, the system should support suitable approximated search techniques to handle large vector sets.

B3 **Adaptive kNN-Join algorithm:** The index should be adaptive to support *B3.1 flexible target sets* and *B3.2 online indexing*.

B3.1 *Flexible target sets:* To support word embedding operations, we want to build one large index over all word vectors provided by an embedding model as well as derived embedding vectors created for text values in the database[10]. However, a target set T of a kNN-Join operation often contains just a small subset of the vectors in the index. In the join query example in Figure 3.5, T is restricted to vectors representing movie titles of a specific genre. This is a much smaller set of vectors compared to the set of all word embedding representations. The kNN-Join algorithm therefore must be adaptive to different target set sizes and should enable fast approximated search. This problem can not be overcome by multiple index structures, since filter criteria can be arbitrary and a large number of index structures leads to a higher demand for memory and longer insertion and update time.

[10]For long text values, the embeddings of tokens are averaged to obtain a representative embedding (see Section 2.2.5).

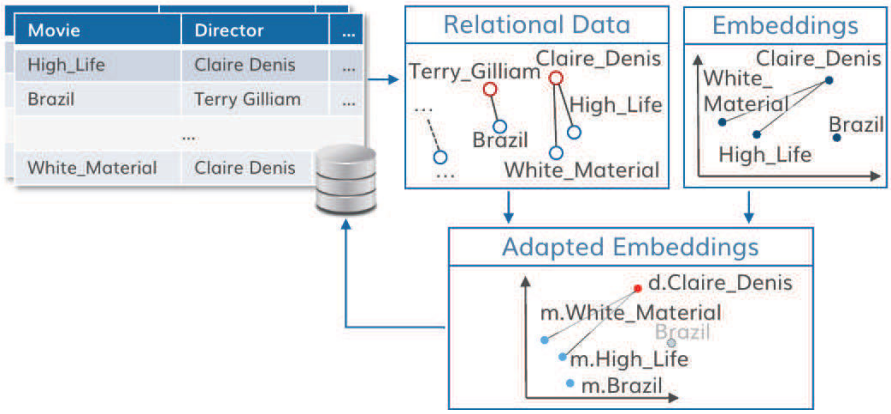


Figure 3.6: Context Adaptation of Embeddings of Text Values in Database

B3.2 *Online Updates:* New text values not present in the word embedding model beforehand can be added to the database during run-time. However, simple techniques that average the embeddings of their tokens [ALM17] provide a convenient way to generate embedding representations for such text value (see Section 2.2.5). To be considered by kNN-Join operations those vectors need to be added to the index structures. Thus, the index structures should allow online-updates.

B4 **Different demands on precision and response time:** Regarding the approximation of the vector similarity, it might be relevant for a user to specify how much the approximated nearest neighbors should agree with the exact values. On the contrary, real-world systems need to comply with certain latency constraints, e.g., for exploratory data processing, fast response times are crucial. Consequently, the approximated kNN-Join should provide features to configure such trade-offs. Providing these tunable trade-offs would also support query execution in an online aggregation manner, i.e., get estimates of a kNN-Join query as soon as the query is issued and steadily refine during its execution.

3.6 CONTEXT ADAPTATION

In text documents, the semantic of a word strongly depend on the context in which the word occurs. This context involves the domain of the text document as well as the surrounding words in the sentence. Similarly, in a relational database, the semantic of the text value depends on the specific domain of the data stored in the DMBS. Moreover, the position of a text value in the database plays an important role, e.g., "Apple" occurring in a column named "Fruits" is differently interpreted when it occurs in a column named "Company". Thus, the semantic of a text value in the database is often more specific then the semantic encoded in a word embedding model. Therefore, our objective is to adjust the embedding representations of text values in the database to better model their specific meaning in the database.

Figure 3.6 sketches the concept of such a context adaptation. The adaptation algorithm expects a database and a pre-trained word embedding model as input. For each text value in the database, a vector representation can be obtained using the word embedding model. Moreover, from the relational database semantic connections between the text

values can be obtained. Both data sources are combined by the adaptation algorithm to obtain an optimized vector representation for each text value. Afterward, these vector representations can be stored in the database system.

To achieve this goal, we start by identifying limitations of word embedding models when applied separately to each text value in the database.

Limitations of Pre-Trained Embedding Models Frequently, word embedding representations do not accurately represent the semantic of textual information in databases. We identified several explanations for this that can be addressed by a context adaption algorithm:

1. Some text values with a context-specific semantic in the database occur in the general domain much more frequently with the broader meaning. Then, this broader semantic overlies the context-specific semantic. Given the movie table, it is known that all entities within the movie column are movies, however, word embedding models misinterpret titles, such as "Brazil" or "Alien" with a different ordinary meaning.

2. Terms in the database occurring infrequently in the general domain can not be modeled accurately by word embedding models. Moreover, many word embedding models have a limited vocabulary and cannot generate embeddings for so-called out-of-vocabulary terms. This is especially unpleasant because it could be circumvented by taking the representations of terms related to the missing term into account.

3. Often, semantic properties of text values are inaccurately represented by a embedding model or missing, even though those properties are implicitly encoded in the word embedding model. For example, it is hard to derive semantic properties of the movie "Brazil" from its embedding representation. However, implicit information about the movie might be encoded in the embedding of its director, e.g., the original language is likely a language the director is able to speak.

Requirements of Context Adaptation To overcome the limitations described above, we aim at developing a novel context adaptation algorithm. To be suitable for relational database systems, this algorithm should support the following requirements:

C1 **Holistic:** The database provides a specific set of relation types modeled by the schema, whereas in word embeddings large amounts of implicit relations are modeled. Both need to be combined.

C2 **Expressiveness:** The adaptation algorithm should model *columnar*, *row-wise*, and *foreign key* relations.

C3 **Online Updateable:** It should be possible to generate representations for text values inserted later into the database. Therefore, the algorithm should be able to generate representations in an online fashion.

C4 **RDBMS Integration:** It should be possible to automatically build context-adjusted representations for text values in a relational database system.

3.7 REQUIREMENTS FOR MODEL RECOMMENDATION

Since word embeddings are essentials for many Machine Learning applications, ML practitioners frequently face the problem of choosing the best embedding model for a specific task. Moreover, for arithmetic word embedding operations, the choice of the word embedding model is important since different embedding models capture different notions of similarity. Thus, the word embedding model should match the notion of similarity fitting to the user's intent. To allow the user to make an informed choice of a word embedding model, our goal is to evaluate the embedding models on a comprehensive evaluation dataset with intrinsic evaluation methods described in Section 2.3. As a result, we obtain an evaluation report, allowing the user to determine proper word embedding models. However, as further detailed in Section 6.1, most of the intrinsic evaluation datasets available are inappropriate for this purpose since they provide just a single quality variable and are either too small, focus on syntactic similarity only, or cover one domain only. Thus our research aims at constructing of suitable evaluation datasets for whom we propose the following requirements:

D1 **Generality:** While a domain-specific evaluation should be possible, one should be able to use the dataset to evaluate any word embedding model. Thus, in its core, the dataset should contain facts that are part of the extended common knowledge.

D2 **Domain Granularity:** To analyze the performance of embedding models on different domain text, the dataset has to be structured into domain-specific categories.

D3 **Continuity:** Many relations are not distinct, e.g. population numbers, state of residence, and occupation depend on a specific point in time. Word embedding models trained on a specific dataset cannot model such time-dependent relationships and common evaluation metrics focus on bilateral relations. Thus, dataset should be restricted to distinct relations and continuously valid knowledge.

D4 **Volume:** To provide a fine-granular view on the performance of word embedding models every domain-specific partition of the dataset should have a certain size since otherwise, accuracy values are not expressive.

D5 **Flexibility:** It should be possible to tailor an evaluation of a word embedding model to a given application domain. Therefore, the structure of the evaluation dataset should provide the flexibility to define different evaluation scopes. Here, a scope may refer to a certain set of objects for which the representation quality should be evaluated. Another scope could be a set of relation types or a certain domain defined by a domain-specific category. To allow these different evaluation scopes, the dataset structure should provide the necessary flexibility.

D6 **Automatic-Generation:** While we aim at providing a comprehensive evaluation dataset, it may not fit for any domain-specific application area. By designing a data-driven construction process, engineers can easily execute this process on any domain-specific table collection of an organization to generate a dataset covering relations of other fields.

3.8 TABULAR EMBEDDING MODELS

Word embedding models are frequently used to represent text values in tabular data (see Section 2.4). However, pre-trained on text documents, those models are rather designed for words in sentences leading to several limitations discuessed below. Based on this observation, we propose to pre-train embedding models on tabular data.

Limitations of Word Embeddings for Tabular Data

1. While word embedding models are trained to represent tokens in a text, algorithms working on tabular data often consider a cell as the smallest structural unit for which they require a single embedding representation. Text values in tabular cells, however, can be rather diverse. There could be abbreviations, single words, multi-words, or even comments spanning over multiple sentences. Moreover, text values in tables frequently contain special signs to format text, which is unusual in text documents. Therefore, word embeddings cannot model them or obtain inaccurate representations for them.
2. Text in tables can be separated into schema and instance data. A text value occurring in the column header of a table has a different semantic when it appears in the table body. However, word embedding models cannot distinguish between those two types of text values.
3. Categorical relations like instance-of relations, which frequently exist between table header and table body, are poorly represented by word embedding models. For example, in the popular pre-trained W2V-GN model (see Table 3.1), the name of the German politician "Angela_Merkel" has a higher cosine similar to "country" (0.18) and "moon" (0.11) than to "person" (only 0.06).

Requirements for Embedding Models To overcome those limitations stated above, we aim at designing an embedding model satisfying the following requirements:

E1 **Flexibility:** Our embedding approach should be flexible enough to obtain a single embedding representation for any text in a cell which can then be used for supervised as well as for unsupervised tasks. Thereby, the model should also be able to generate embeddings for text values that do not appear in the training corpus, which is especially important because of the large variety of text values in tables. In this way, it should be possible to apply the pre-trained model to different potentially much smaller tabular datasets.

E2 **Schema Awareness:** It should differentiate between text that represents schema information in a table and text that represents instance data.

E3 **Modeling of Tabular Relations:** The model should be trained on tabular data instead of text documents. Thereby, the model should learn from tabular relations like row-wise relations and relations between schema and instance terms, as they exist between the header terms and the body of the table.

4

MANAGEMENT OF EMBEDDING REPRESENTATIONS IN DATABASE SYSTEMS

In this chapter, we investigate techniques to manage word embedding representations in a database system and and propose a concept to implement novel word embedding operations according to the objectives A and B in Section 3.3. The results of this research have been published in several publications [Gün18, GTLY19, GTL19a].

In Section 4.1, we introduce the architecture of FREDDY (Fast Word Embeddings in Database Systems) [Gün18], a prototype of an extension for PostgreSQL to integrate word embedding operations. The code is published as an open-source project on Github[1]. In addition, a demo of the system has been published in [GTLY19], which provides a Web interface for FREDDY[2] and is described in detail in Section 4.1.4. To provide the desired efficiency, we review related work on fast approximated nearest neighbor search in Section 4.2, discuss their applicability for word embedding database systems in Section 4.3, investigate kNN search features for database systems in Section 4.4, and introduce a novel algorithm for approximated kNN-Joins (ANN-Joins) [GTL19a] in Section 4.5, which we integrate into FREDDY.

4.1 INTEGRATION OF OPERATIONS IN AN RDBMS

We choose to integrate the desired additional functionality into PostgreSQL because it is one of the most popular open-source database systems[3], and it provides utilities to build extensions. Section 4.1.1 describes the system architecture. In Section 4.1.2, we discuss how to store word embedding representations in the database. It follows a detailed description of additional functions, which we implemented in PostgreSQL to enable word embedding functionality in Section 4.1.3. Finally, we describe the Web interface in Section 4.1.4.

4.1.1 System Architecture

PostgreSQL can dynamically load extensions with object code in the form of shared libraries[4]. This enables us to develop additional functions which can be executed effi-

[1] https://github.com/guenthermi/postgres-word2vec (Access: 04/27/2021)
[2] https://github.com/z-yan/freddyDemo (Access: 04/27/2021)
[3] It is the 4th most popular DBMS according to https://db-engines.com/en/ranking. (Access: 04/27/21)
[4] https://www.postgresql.org/docs/13/extend-how.html (Access: 04/27/21)

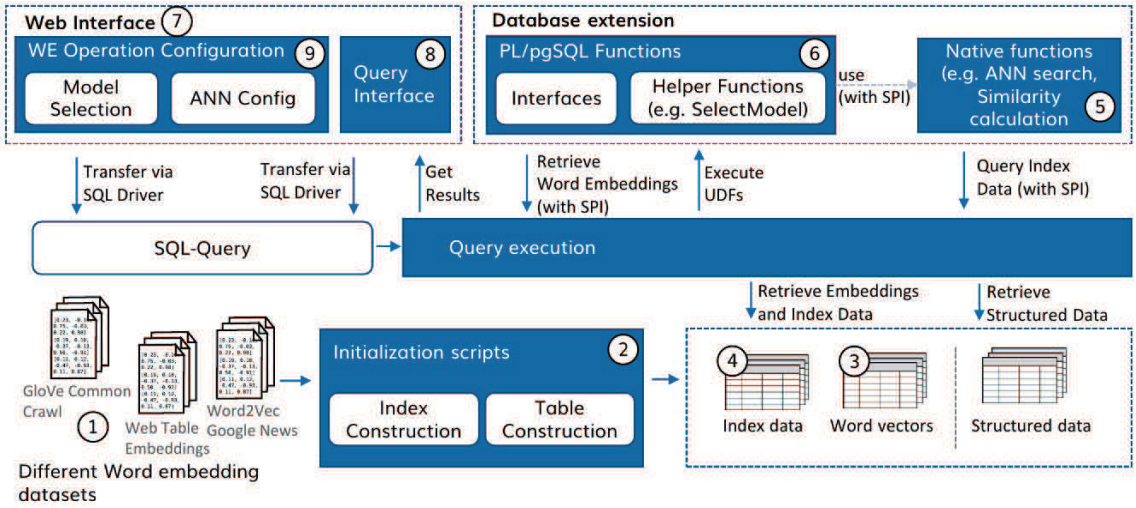


Figure 4.1: FREDDY System Overview

ciently by the database system. FREDDY's system architecture is sketched in Figure 4.1. Multiple word embedding datasets (1) can be added to the system. Initialization scripts (2) create new relations (3) for word vectors in those datasets and index structures (4) to enable fast approximated kNN queries. For each word embedding model, the scripts construct separate tables. However, all vector representations derived from a specific embedding model are stored in the same tables. Section 4.1.2 exposes the details of the storage formats. To exploit the capabilities of these word embeddings within SQL, we implemented User-Defined Functions (UDFs) for the operations described in Section 3.4 that operate on the word embedding and index relations. In this way, Requirement A1 of Section 3.4 is satisfied. Those operations can either be executed on all embedding representations stored in the table dedicated to the currently selected word embedding model or on a restricted set of representations. Additional UDFs serve as helper functions, e.g., to select a word embedding model for the execution of the operations. Section 4.1.3 provides a comprehensive overview of the functions. The extension allows an exact calculation of search functions like kNN, but also provides the possibility to perform them in an approximated manner to enable the execution on large input sets and tables. The UDFs for similarity calculations and search operations (5) are implemented in C, whereas performance uncritical helper functions and interfaces (6) are realized via the procedural script language PL/pgSQL. By using the PostgreSQL Server Programming Interface (SPI), the UDFs can run SQL commands inside the functions, e.g., to access the word vectors and index structures. All UDFs are bundled into a PostgreSQL extension.

To increase the usability of the system, we developed a Web application (7) to provide an interface for our system. This application offers a convenient SQL interface for PostgreSQL (8). Besides, a graphical user interface supports the user to utilize the additional functionality (9) provided by FREDDY. FREDDY supplies helper functions to switch between word embedding models used by the novel operations and configure the hyperparameters, e.g., for ANN search. To simplify this, the Web application provides widgets for those purposes and thereby satisfies Requirement A5. By employing the widgets, the application constructs SQL queries including the helper functions, and executes them on PostgreSQL via an SQL driver. In Section 4.1.4, we give a detailed overview of the functionality of the Web Interface.

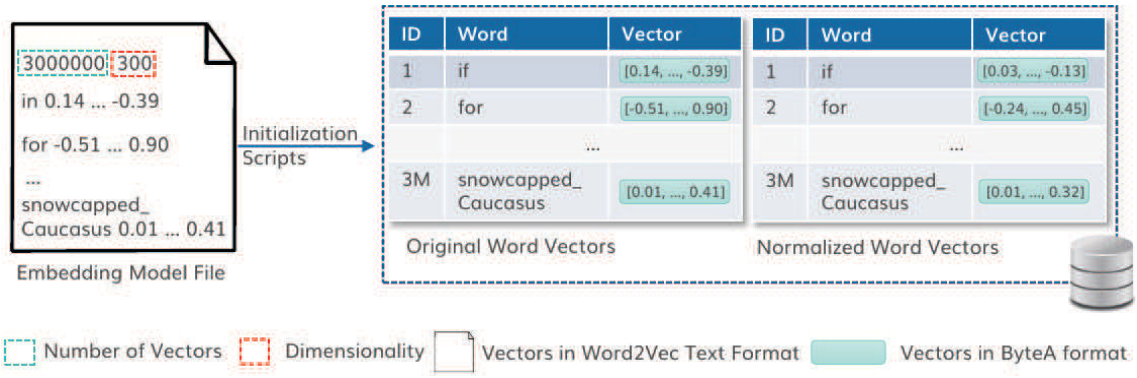


Figure 4.2: Storage Formats for Word Embeddings

4.1.2 Storage Formats

Figure 4.2 shows the different storage formats of word embedding representations. Static word embedding representations are usually stored either in the text format shown on the left in Figure 4.2 or in a similar structure in binary representation to save space. Both formats can be converted into each other, e.g., by using the Gensim library [RS10]. The initialization scripts create two tables for a vector file in the text format. Both tables contain an id column, a column storing the text value, and a column for the corresponding vectors. The first table stores the original vectors, and the second table contains a normalized version of the vectors to efficiently calculate the cosine similarity between two vectors. To efficiently retrieve word vectors, index structures are built over the id and the word column. Usually, either hash indexes or B-Tree indexes are used.

Although PostgreSQL provides array types, we decide to store the vectors as byte arrays with the type `ByteA` because it allows a more memory-efficient representation. All vectors are stored with 32-bit floating-point values so that each vector requires $4 * d + c$ bytes, where d denotes the dimensionality of the vector and c is a constant overhead of up to 4 bytes[5]. PostgreSQL allows compressing `ByteA` values with a technique called TOAST (The Overhead-Attribute Storage Technique)[6]. However, this is usually not effective for word embedding vectors and therefore not applied in most cases.

If text values not present in the word embedding model are added to the database, e.g., because they consist of multiple tokens, new embedding representations can be added for them using the *insert_batch* function. Those embedding representations are added to both embedding tables (see Figure 4.2), as well as to the tables of index structures described in Section 4.5.1. As mentioned in Section 4.1.1, we store all embedding representations of text values in the database in the same tables and do not create a separate embedding table for each text column. In this way, a lot of storage can be saved because only one vector needs to be stored for the same text value occurring in multiple columns. This is especially relevant because the embedding vectors and their index entries usually require much more memory than the actual text values. Moreover, several of the word embedding operations introduced in Section 3.4 apply search operations on all word embeddings, e.g., for data exploration, or can be applied to embeddings of text values in multiple tables. To efficiently utilize index structures in those cases, the indexes need to be built over all embedding vectors.

[5] https://www.postgresql.org/docs/13/datatype-binary.html (Access: 04/28/21)
[6] https://www.postgresql.org/docs/13/storage-toast.html (Access 28/04/21)

Binary Word Embeddings In addition to common static word embeddings, FREDDY also supports binary word vectors trained with word2bits [Lam18] already described in Section 2.2.7. Here each vector corresponds to a bit vector where each bit corresponds to $\frac{1}{3}$ for a value 1 or $-\frac{1}{3}$ for value 0. Storing those vectors as floating-point values is inefficient. Fortunately, the TOAST compression reduces the size significantly[7], however, a binary representation is still much more efficient. Therefore, we partition the original bit vector in chunks of 64 bit and represent them as an array of unsigned 64 integers and store this with the `bytea` type. To enable word embedding operations on those vectors, specific implementations are provided (see Section 4.5.9).

4.1.3 User-Defined Functions

The database extension consists of implementations of performance-critical functions implemented in C and a PL/pgSQL script. The C implementations are compiled to native code packed into a shared library. When the extension is added to the PostgreSQL server, the PL/pgSQL script is executed. This script adds bindings to the native functions implemented in the shared library and implements several additional UDFs in PL/pgSQL. Those include higher-level word embedding operations using the native functions, helper functions, and multiple interfaces to each word embedding operation. Moreover, the script executes code to initializes the extension.

Initialization: During the initialization, the *init* function is executed to set a default word embedding model. This function gets as arguments the names of tables created for the word embedding model and the tables providing index structures for them (see Section 4.5.1) Afterward, the word embedding operations can retrieve the table names of the selected model by executing specific UDFs. This enables model independence described in Requirement A4 in Section 3.4. Moreover, hyperparameters for the ANN search are set.

Word Embedding Operations: We implement UDFs for the functions already described in Section 3.4 into FREDDY. To efficiently calculate similarity values, a native function calculates the dot product between two normalized vectors, which can be obtained from the table of normalized word vectors (see Figure 4.1.2). All the higher-level word embedding operations are either implemented in PL/pgSQL and use the cosine similarity operation, or native functions implement them. For more efficient approximated implementations of the word embedding operations, we focus on developing a novel adaptive algorithm to efficiently execute the ANN-Join operation (see Section 4.5) that fulfills the requirements stated in Section 3.5. All other word embedding operations, besides the simple similarity function, can be implemented efficiently via this ANN-Join operation:

- *kNN:* The implementation of ANN and ANN_in queries via ANN-Joins is trivial. Here, an ANN-Join with a single vector in the query set R is used. The target set T is restricted for ANN_in queries, otherwise, all vectors are added to the target set.
- *Grouping:* A grouping operation is an ANN-Join with $k = 1$ where the target T set usually consists only of a few target vectors.

[7] For 800-dimensional vectors, we observe a compression ratio between 6 and 8.

- *Clustering:* To implement a cluster operation based on the k-means algorithm, the function iteratively executes ANN-Joins with a query set R corresponding to the set of cluster centroids, $k = |T|$, and a target set T corresponding to the set of input vectors. To obtain an assignment of each input vector to a cluster, the results of the join are ordered by the similarity values between query and target vector[8]. Afterward, the function uses this result list to assign each input vector corresponding to a target vector to its most similar centroid corresponding to a query vector. For the first join, the centroids are initialized with randomly selected input vectors. After each iteration, the function recalculates the centroids. To efficiently do this, the centroids are calculated based on a sample of vectors in each cluster. After the last iteration, the function returns the assignment of the ANN-Join.

- *Analogies:* For analogy operations, the 3COSADD method, explained in Section 2.2.5, can be used. The other analogy operations described in Section 2.2.5 are also implemented in UDFs. However, here no optimization is applied. To optimize the efficiency of 3COSADD, a vector is calculated based on the three input vectors according to Equation (2.9). Then, this vector is normalized, and an ANN-Join with $k = 1$ and a target set T holding all word vectors is applied to retrieve the results. For the analogy_in function, the target set T contains only the set of vectors provided to the analogy function.

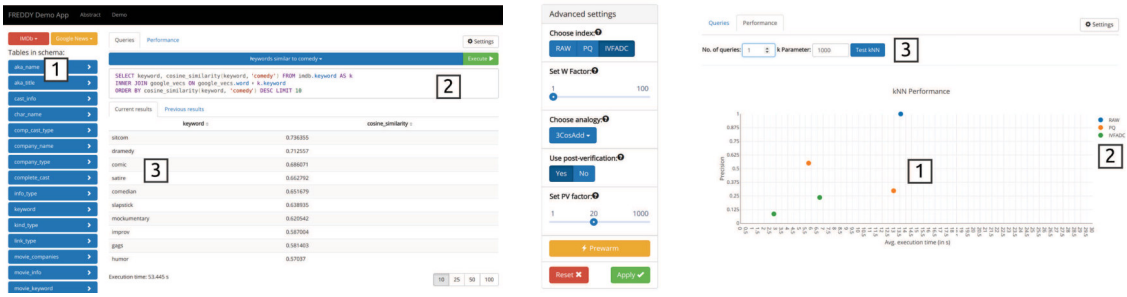
To efficiently execute word embedding operations trained with a word2bits model, we use different implementations for cosine similarity and kNN-Join[9] described in Section 4.5.9.

Helper Functions: There are several functions to read out and adjust the hyperparameters set during the initialization. Similarly, helper functions enable the user to read out and switch the implementation underlying the execution of an operation like kNN, e.g., to change from an exact algorithm to a faster approximate algorithm.

The *insert_batch(varchar[])* function adds representations of text values not present in the word embedding model. Therefore, it uses the averaging method [ALM17] explained in Section 2.2.5.

Furthermore, several helper functions are provided to convert arrays between the `ByteA` type and PostgreSQL array types. Those are implemented as native functions.

Interfaces: The user can provide embedding representations to the word embedding operations in three different formats: (1) vectors in the `ByteA` format, (2) text values referring to embedding representations present in the vector tables, and (3) ids in the vector tables. The native word embedding function either accept vector representations or ids. For increased usability interface functions implemented in PL/pgSQL support the alternative formats (1) - (3), read out the required data from the vector tables, and call the native functions to execute the operation. In this way, the system complies with Requirement A3.



(a) Query Interface (b) Configuration (c) Search Analysis View

Figure 4.3: The web-interface of FREDDY

4.1.4 Web Application

A Web application provides a user-friendly interface for FREDDY. A screencast is provided on the project website[10]. The application consists of a Web client for the database with additional functionality to support word embedding operations. The user can enter a query, or select and edit a pre-defined queries. The application allows the user to switch between different databases and word embedding models. To gain detailed insights on how the different index structures and search functions perform and how their parameters affect result quality and query performance, we provide several widgets to select and adjust them. On a second view, the user can evaluate the influence of different index structures and their search parameters.

Query Interface View The query interface is displayed in Figure 4.3a. The user can choose between different database schemes (1). For demonstration purposes, we imported a database with movie data, a DBLP[11] database, and a music database with data from Discogs[12]. A drop-down menu enables the user to select different word embedding datasets from a selection of inserted embedding models. One can add models like W2V-GN, GV-CC, and W2B-W of Table 3.1. Executing the same query multiple times using different word embeddings leads to different result sets. In general, it is recommended to choose a word embedding dataset pre-trained on a related topic according to the database schema. In the text field at (2), the query can be created manually, or the users get inspired by one of the pre-defined example queries provided by the drop-down menu above. If a query is executed, its result and the response time appear at (3). The demonstrator also keeps track of the previous query and its result. They can be retrieved and compared with the current ones using the tab menu above the result table. In a sidebar (see Figure 4.3b), the users can choose between the index structures for similarity search and different analogy query types.

Performance View In a second view illustrated in Figure 4.3c, the demo user can perform time and precision measurements for kNN and analogy queries using different configurations and compare the results by employing various plots (1). Visual features, e.g.,

[8] Although the similarity values are not in the result set according to the kNN-Join definition in Section 3.5, those values need to be calculated in any case and therefore can be returned by the kNN-Join operation.

[9] Exact methods for word2bits operations are more efficient. Therefore, an approximation is not necessary.

[10] https://wwwdb.inf.tu-dresden.de/research-projects/freddy/ (Access: 04/28/21)

[11] https://dblp.uni-trier.de (Access: 08/26/21)

[12] https://www.discogs.com (Access: 08/26/21)

color, size, etc., encode the index and search parameters. The notations are declared in the legend at (2). To obtain reliable measurements, the queries are executed multiple times, and the average values for the response time and the precision are obtained. At (3), the number of queries that should be executed and the neighborhood k are specified. The user can configure the search function in the sidebar (see Figure 4.3b), just as in the query view.

4.2 NEAREST NEIGHBOR SEARCH

In the following, we investigate techniques to efficiently implement nearest neighbor search. The kNN search problem defined in Section 3.5 and approaches for efficiently solving it have been extensively studied in the literature. There is not one specific algorithm that fits all applications. The efficiency of kNN search algorithms can strongly depend on the data characteristics of the vector datasets. Moreover, index structures for kNN search differ significantly in their memory consumption, index construct time, and the index operations they provide, e.g., not all indices allow online updates.

A naïve algorithm performs a scan over all vectors in the target set, calculates the distance to each vector, and selects the k vectors with the lowest distance values. This algorithm has a linear time complexity of $\mathcal{O}(|T|\cdot d)$ for the size of the target set $|T|$ and the dimensionality d. To perform a kNN-Join between R and T, one can execute a kNN search for every query vector $r \in R$. Here, the time complexity $\mathcal{O}(|R|\cdot|T|\cdot d)$ further depends on $|R|$. Since word embedding datasets usually consists of millions of vectors with hundreds of dimensions, calculating all distance values is inappropriate for real-time queries.

To efficiently compute the nearest neighbors, several index structures have been proposed. In general, due to the curse of dimensionality, exact methods like [FN75, FBF77, Nav02] do not effectively reduce the run-time of kNN search queries [MY18]. The dimensionality of popular pre-trained word embedding models is sufficiently high that this effect occurs (see Section 3.2). The most popular approaches for approximated nearest neighbor search (ANN search) can be categorized into techniques using *tree structures* to organize vectors, *proximity graphs*, methods base on *locality-sensitive hashing*, and techniques base on *quantization*.

4.2.1 Tree-based Methods

Several methods [FB74, Ben75, Gut84] have been proposed to build tree-structured indexes for multi-dimensional data points. One of the most popular hierarchical index structures is the *k-d tree* [Ben75, FBF77]. The k-d tree is a binary tree storing k-dimensional vectors associated with its leaf nodes. All non-leaf nodes are associated with a partition of the vector space where the root node represents the whole vector space. All non-leaf nodes define a hyperplane that is perpendicular to an axis of the vector space and divides its partition into two partitions associated with its two successors nodes as in the example in Figure 4.4. The hyperplane is defined by a key a referring to the axis and a threshold value v. All vectors in the partition of the parent node with a value greater than v at dimension a are assigned to the partition of the first child node and all other nodes are assigned to the partition of the second child node. Several different methods for constructing a k-d tree are possible. Given all vectors in advance, one can use the approach of [FBF77] to construct an optimal k-d tree for searching nearest neighbors.

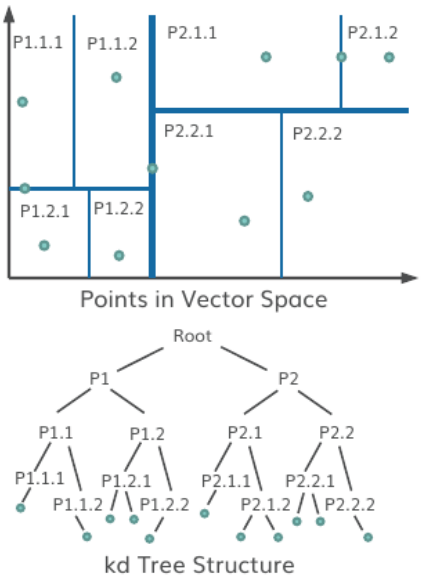


Figure 4.4: k-d tree

Algorithm 4.1: KDSEARCH(**r**,$n = root$,$C = \emptyset$)

Input: query vector **r**, node n, candidate list C
Output: k nearest neighbors

```
1  if n is Leaf? then
2  |  C = UPDATEKNN(r, n, C);
3  |  return C;
4  else
5  |  n_near, n_far = GETSUCCESSORS(n, q);
6  |  KDSEARCH(n_near, r, C);
7  |  if BOUNDSOVERLAPTEST(n_far, C) then
8  |  |  return C;
9  |  else
10 |  |  C = KDSEARCH(n_far, r, C);
11 |  end
12 end
13 if BOUNDSWITHINTEST(n, C) then
14 |  return C and quit ;        // terminate directly
15 else
16 |  return C ;          // continue with parent node
17 end
```

Figure 4.5: k-d tree Search Algorithm

k-d tree Search Algorithm: To find the k nearest neighbors, one can use the recursive search algorithm shown in Figure 4.5. This algorithm maintains a list C of k already investigated candidate vectors with the lowest distance to the query vector **r**. Each recursive step processes a node n of the graph where the first step examines the root node. In each step, it is checked if the current node is a leaf node. For a leaf node, it is determined if the vector associated with it is a candidate for the k nearest neighbor by calculating its distance to the query. If the node is a non-leaf node, a new recursive step is executed for the successor node n_{near} holding vectors on the same side of the hyperplane. After this new recursive step is executed, the algorithm checks if it is possible that the second successor node n_{far} holds vectors closer to the query than the candidate list. If this is the case, this node is investigated in another recursive step. Otherwise the algorithm terminates.

To check if a partition of the vector space associated with a node needs to be investigated a *bounds-overlap-ball test* is done. Thereby, the algorithm calculates the ball centered around the query vector **r** with the radius equal to the distance between the query and the farthest vector in the candidate list. If this ball overlaps, it is necessary to investigate the partition of the node. Also, a *ball-within-bound test* can be performed after each recursive step to check whether the ball lies in the partition of the node and the algorithm can terminate.

While this search algorithm only provides effective performance improvements on low-dimensional data, for high-dimensional data algorithms for ANN search have been proposed. The popular approach of [SAH08] utilizes a combination of multiple k-d trees for a very efficient approximation of the nearest neighbors which is implemented by [ML14] in the FLANN framework (Fast Library for Approximate Nearest Neighbors)[13].

[13] https://github.com/mariusmuja/flann (Acess: 04/01/21)

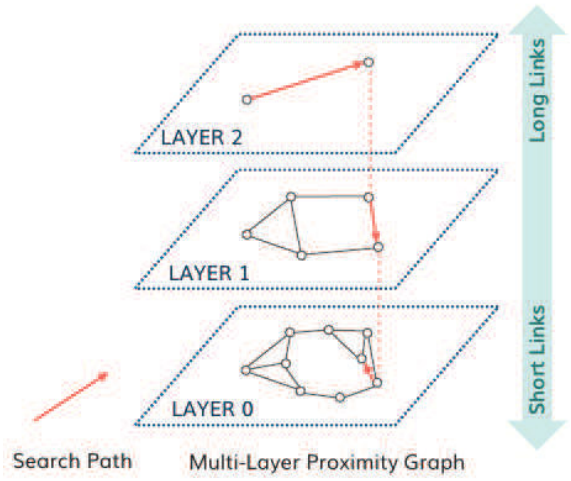


Figure 4.6: HNSW Graph and Search Algorithm

R-Tree: To construct a near-optimal k-d tree, the distribution of the data needs to be known beforehand. Thus, updating k-d trees can lead to lower performance improvements. Moreover, k-d trees are not able to effectively index geometric objects like rectangles and polygons. To overcome those limitations, the *R-Tree* [Gut84] and several variants of it [SRF87, BKSS90, BKK96] were developed. The idea behind the R-Tree is to organize spatial objects in a hierarchy of bounding boxes. Objects are associated with leaf nodes. For each node, a minimal bounding box of the child nodes (or objects in the case of leaf nodes) is stored. To insert objects, the tree is traversed to the best fitting leaf nodes[14] and the object is added to the node. If the node overflows, i.e., the maximal number of objects is reached, the insertion algorithm splits the node and adds both splits to the parent node. This can also cause further overflows in the intermediate nodes leading to further splits. R-Trees can efficiently solve k nearest neighbor queries [RKV95] as well as range queries.

4.2.2 Proximity Graphs

Recently, techniques for ANN search based on proximity graphs have become very popular. Many of the state-of-the-art ANN techniques use such graphs [ABF20]. A proximity graph relates nodes that are close according to a certain distance measure. One form of a proximity graph is the k nearest neighbor graph (kNN graph). The kNN graph is a directed graph that represents each spatial point by a node and relates each node to its k nearest neighbors. For low-dimensional data, [PCFN06] proposes an algorithm to efficiently construct kNN graphs. However, due to the curse of dimensionality, it has a nearly quadratic complexity for high-dimensional spaces. Based on the kNN graph, it is possible to directly return kNN search results for query vectors in the graph. Thus, the time complexity is $\mathcal{O}(1)$. For query vectors not in the index, search algorithms are proposed to navigate via a *greedy search* from an initial node through the kNN graph towards nodes with small distance values to the query vector [HAYSZ11]. To find a starting point, it is common to choose a node randomly or a separate index structure is used, e.g., in [AM93] an additional k-d tree is constructed and [WWZ+13] employs product quantization for this purpose.

[14] Here, different heuristics are possible.

Hierarchical Navigable Small World Graphs (HNSW): One of the most popular techniques in this category on ANN search methods is based on *hierarchical navigable small world graphs (HNSW)* [MY18]. To efficiently search, HNSW constructs a multi-layer proximity graph depicted in Figure 4.6. This graph is constructed by an insert algorithm that incrementally adds nodes for each target vector. In such a graph a lower layer always contains all nodes from the upper layers. The first node inserted on the top layer of the graph is the initial node for the search also called the *enter-point*. The enter-point is the same for all search operations independent of the query vector.

Insertion Algorithm: First, the insertion algorithm selects a maximum layer L_{max} for the new node determined by an exponentially decaying probability distribution. Then, it traverses the graph with a greedy algorithm starting from the *enter-point* on the highest layer until layer L_{max} is reached. This greedy algorithm[15] aims at selecting nodes of target vectors close to the query vector in the current layer. Then, it investigates the next layer in the same manner starting from the closest point determined on the previous layer. When the maximum layer L_{max} for the new node is reached, the actual inserting process starts. Thereby, a similar greedy algorithm is executed. It differs from the former one in the way that it obtains multiple approximated nearest neighbors on each layer which serve as enter-points on the next layer. Moreover, a node for the new vector is added on each layer, as well as a specific number of edges to close neighbors. Afterward, the set of edges of each neighbor is reduced to a maximal number of neighbors M_{max} to prevent nodes to hold too many connections.

Search Algorithm: The search algorithm proposed by [MY18] resembles the insertion algorithm for a node on layer 0. It greedily searches for close neighbors on each layer until it reaches layer 0. Here, it uses the closest node in layer 1 as the entry point to search for $ef > k$ close neighbors. Afterward, the k vectors with the lowest distance values are returned.

Based on the benchmark results of [ABF20], this technique provides the best trade-off between accuracy and search run-time while index construction time and the size of the index structures required to achieve this performance are comparably high. The evaluation of recently developed novel graph-based approaches [IM18] and quantization-based techniques [GSL+20] shows that these techniques achieve comparable or superior performance trade-offs[16].

4.2.3 Locality-Sensitive Hashing

Conventional hashing functions used for data structures are designed to equally likely hash an input value to an integer representing a storage bucket id independent from the input [CLRS09]. In contrast, locality-sensitive hashing (LSH), first proposed by [IM98, GIM99], tries to hash similar inputs to similar hash values.

Index Construction: Figure 4.7 visualizes the indexing process of vectors with an LSH technique later used for ANN search. Thereby, several LSH hash functions $H_1, \ldots, H_l$ are applied to each input vector $\mathbf{v_1}, \ldots, \mathbf{v_n}$. Usually, those hash functions generate a bit-vector of length m for the original floating-point vector where similar vectors have a high probability to obtain the same bit-vector. Afterward, a conventional hash function is applied to each bit-vector to get an integer value $x \in \{1, \ldots, S\}$ for each of them. Those integer values constitute indexes in a hash table with S buckets. The hash tables store for each vector a reference at the position of its hash value x. In this way, LSH provides a tool for inverted indexing where the location of the object in the form of the bucket id can be derived form its content (the original vector).

[15] For a detailed description of the algorithm, we refer to the original paper [MY18].

[16] http://ann-benchmarks.com/ (Access: 04/09/2021)

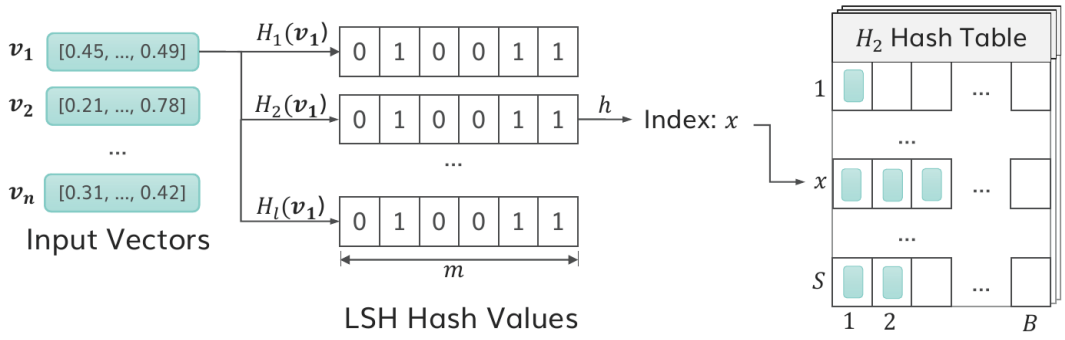


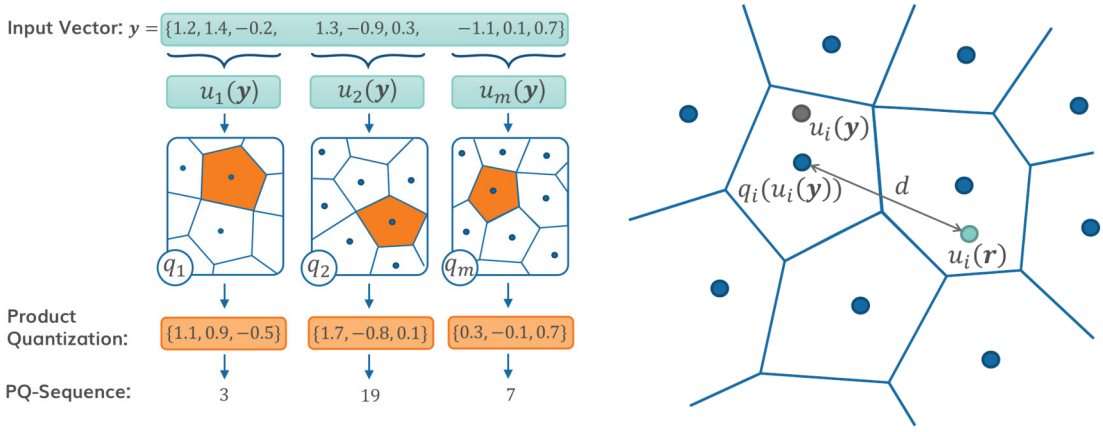
Figure 4.7: Index Vectors with locality-sensitive Hashing (LSH)

Design of the Hash Functions: Depending on the vector space and the distance measure different hash functions can be used. In the original approach [GIM99], hash functions are designed for d-dimensional integer vectors and the Manhattan distance (L1 distance). The authors argue that the L1 distance correlates with the Euclidean distance. Moreover, to handle floating-points, one can transform them to integer values by scaling them with an arbitrarily high factor and rounding them afterward. To calculate a hash value of a d-dimensional vector $\mathbf{v}$, all d integer values are transformed into unary encodings of length C (filled up with leading zeros), where C constitutes the highest integer value in the dataset. Then, the unary encodings of a vector are concatenated to a bit-vector of length dC. For each LSH hash function $H_1, \ldots, H_l$, a subset $A \in \{A' \in \mathcal{P}([dC]) \mid |A'| = m\}$ of m dimensions is chosen. An LSH hash function selects the values at those dimensions and concatenates them to obtain a bit-vector for its input.

By changing the definition of the hash function, other vector spaces and distance measures can be supported. In [Cha02], hash functions are designed for the cosine distance. In [DIIM04], an approach specifically for Euclidean distances is proposed. Recently, [AR15] proposes an approach for data-dependent hashing to exploit the data characteristics of a specific dataset.

ANN Search with LSH Index: To determine approximated nearest neighbors, the search algorithm applies all hash functions $H_1, \ldots, H_l$ to the query to obtain l bit-vectors like this was done in the indexing process. Then, the conventional hash function transforms those bit-vectors to l bucket indexes and all vectors in the buckets are retrieved. Those vectors constitute candidates for the nearest neighbors. In the original paper [GIM99], the authors propose to retrieve vectors only until a maximum number M is reached. Afterward, the search algorithm calculates for each candidate its distance to the query vector and returns the k vectors with the lowest distances.

A limitation of this algorithm is that it is not possible to increase the precision of the search without reconstructing the index with other parameters. Furthermore, increasing the number l of hash function leads to more hash tables. Thus, this results in higher storage requirements. To overcome this, [LJW+07] proposes the multi-probe LSH algorithm. Here, the search algorithm retrieves vectors from more than one buckets per hash function. To increase recall from each LSH hash value, it derives from the hash value multiple slightly modified hashes leading to an extended set of buckets. Since LSH hashes tend to be similar for similar vectors, those buckets contain similar vectors with high probability.



(a) Generating PQ sequences

(b) Asymmetric Distance Calculation

Figure 4.8: Product Quantization

4.2.4 Quantization Techniques

Several kNN search techniques employ forms of *vector quantization*, a method to transform multi-dimensional data points in approximated compact representations [Gra84]. It is the basis of algorithms for compressing vector data, allows the calculation of approximated distances, and enables the construction of spatial indexes.

Quantization Function Vector quantization can be implemented by a quantization function $q : \mathbb{R}^d \to \{1, \ldots, |C|\}$ which assigns a vector $\mathbf{y} \in \mathbb{R}^d$ to the closest centroid vector $\mathbf{c_j} \in C$ of a fixed finite set $C \subset \mathbb{R}^d$. There are different ways to obtain a quantization function specified by the centroid set C and a distance function d. As a distance function, usually, the Euclidean distance is used. Set C should be selected so that the distortion is minimal. The *k-means* algorithm is commonly used to achieve this goal for a given number of centroids $|C|$. Here, the Euclidean distances between each vector $\mathbf{y}$ and its centroid $\mathbf{c_j}$ equivalent to the mean squared error of the differences in each dimension is minimized. For efficient ANN search according to the cosine distance, [GSL+20] propose an alternative algorithm to determining good sets of centroids. The quantization can be visualized as a Voronoi diagram like the one shown in Figure 4.8b. It displays the centroids (blue points) and the volumes assigned to them by the quantization function bounded by the blue lines. The Voronoi diagram itself allows performing a nearest neighbor search in 2-dimensional vector space in logarithmic time [Sha75]. However, this becomes inefficient with growing dimensionality [AMN+98]. To lossy compress vector data, each vector can be replaced with an id of the centroid vector assigned to it by the quantization function. This reduces each vector to a single number requiring $\lceil ld|C| \rceil$ bits to be stored. To index a dataset, the vectors are stored together with their quantization ids and a conventional index structure is constructed for the quantization ids.
Inverted Indexing: For searching approximated nearest neighbors, one can calculate the nearest α centroids to the query vector, retrieve all vectors with the according centroid ids as candidate solutions, and calculate their distances to the query vector to obtain close vectors.

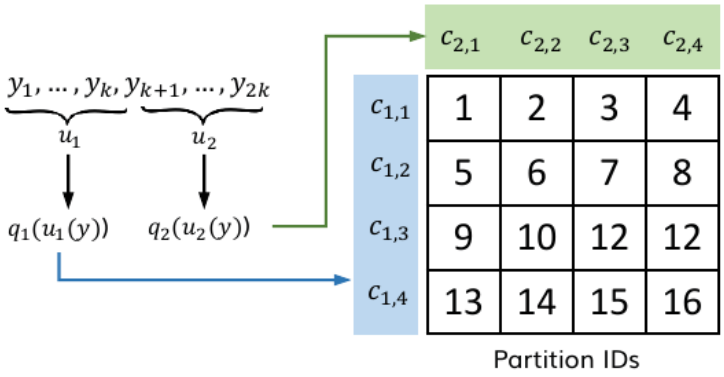


Figure 4.9: Inverted Multi Index

Product Quantization A simple vector quantization approach might lead to a quite inaccurate representation of the vector dataset. For a more precise representation, huge numbers of centroids would be necessary that are impossible to process or even to store. For this reason, *product quantization* [JDS11] applies multiple quantizers on m subvectors $u_1(\mathbf{y}), \dots, u_m(\mathbf{y})$ of the original vector $\mathbf{y} \in \mathbb{R}^d$ (see Figure 4.8a.). Those quantizers are defined by quantization functions $q_1, \dots, q_m$ with $q_i : \mathbb{R}^{d'} \to \{1, \dots, |C_i|\}$. Typically, the cardinalities $|C_1|, \dots, |C_m|$ are equal and $d' = d/m$ for each quantization function. The product quantization is the sequence of centroids obtained by that process.

$$\underbrace{y_1, \dots y_d}_{u_1(\mathbf{y})}, \dots, \underbrace{y_{(D-d)+1}, \dots y_D}_{u_m(\mathbf{y})} \to q_1(u_1(\mathbf{y})), \dots, q_m(u_m(\mathbf{y})) \tag{4.1}$$

Using a dictionary denoted as the *codebook*, the sequence of centroid vectors can be compactly represented as a sequence of centroid ids.

kNN-Search with PQ-Index Product quantization sequences can be utilized to accelerate the calculation of nearest neighbors by providing a fast way to compute approximated squared distances. Approximated square distances between a query vector $\mathbf{r}$ and a vector $\mathbf{y}$ for which a product quantization sequence is available can be calculated by Equation (4.2):

$$\hat{d}^2(\mathbf{r}, \mathbf{y}) = \sum_{i=1}^{m} d(u_i(\mathbf{r}), q_i(u_i(\mathbf{y})))^2 \tag{4.2}$$

The squared distances $d(u_i(\mathbf{r}), q_i(u_i(\mathbf{y})))^2$ have to be precomputed at the beginning of the search process. For every subvector $u_i(\mathbf{r})$, there are $|C_i|$ distance values to calculate since $q_i(u_i(\mathbf{y}))$ can be any value of C_i. The distance measure is denoted as asymmetric by [JDS11] since it is defined between quantized and non-quantized vectors as visualized in Figure 4.8b. Despite the effort of the preprocessing, the technique reduces the computational costs since the number of index entries in large vector datasets is much higher than $m \cdot |C_i|$, the number of those squared distances. Furthermore, the retrieval of the compact product quantization sequences is faster than retrieving the raw vectors. To further speed up the ANN search, [JDS11] proposes to use standard vector quantization as described above to build an inverted index for the product quantization sequences.

Inverted Multi-Index A simple inverted index based on quantization could be created by clustering all possible target vectors T_I into n distinct partitions $P_1 \dot{\cup} \ldots \dot{\cup} P_n$ that correspond to the Voronoi cells of the centroids $\mathbf{c_1}, \ldots, \mathbf{c_n}$. To determine the partitions in which to search for a query $\mathbf{r}$, one has to calculate all the distances $d(\mathbf{r}, \mathbf{c_1}), \ldots, d(\mathbf{r}, \mathbf{c_n})$. However, this could be time-consuming, especially for kNN-Joins with a large query set R. To solve this problem, [BL12] propose to use product quantization to obtain a large number of partitions with low computational effort. Suppose the product quantization sequences which serve as labels for the partitions consist of two centroid indexes $c_1, c_2 \in \{1, \ldots, n\}$. There are n^2 partitions (see Figure 4.9). However, to determine the nearest clusters, one has to calculate only the $2 \cdot n$ square distances between the subvectors of the query vector centroids stored in a codebook.

4.3 APPLICABILITY OF ANN TECHNIQUES FOR WORD EMBEDDING KNN-JOINS

To efficiently execute kNN-Join operations on word vectors in database systems, a flexible index structure is required. Accordingly, the goal is to design an index structure satisfying the requirements stated in Section 3.5. In the following, we analyze the applicability of the ANN techniques summarized above in Section 4.2.

All techniques described in Section 4.2 aim at minimizing memory access to select useful candidate vectors according to Requirement B1. As of April 2021, the best trade-offs between query time and accuracy on a popular word embedding ANN benchmark proposed in [ABF20] are achieved by techniques based on graphs and product quantization[17]. The implementation of batch-wise nearest neighbor search can further account for this requirement. However, this rather depends on the kNN-Join implementation and is relatively independent of the indexing technique itself. In contrast, the feasibility of the other three requirements strongly depends on the index algorithm.

Tree-based Indexes: Tree-structured indexes are mainly proposed for low-dimensional data. However, ANN search based on multiple k-d trees has been proposed [SAH08] to support high-dimensional data as stated in Requirement B2. Tree-based indexes are inappropriate for flexible target sets demanded by Requirement B3.1 because the whole tree needs to be traversed first to obtain candidate vectors. Only after that one can check the filter constraint. Online updates mentioned in Requirement B3.2 are possible. However, the performance of the k-d tree might suffer from too many insertions. To increase the accuracy one can easily retrieve a higher number of candidate vectors from the index close to the query. In this way, the search algorithm can adapt to different demands on accuracy and run-time (Requirement B4)

Graph-based Indexes: Although the construction of optimal kNN graphs is infeasible for large datasets of high-dimensional vectors, proximity graphs are commonly used for ANN on high-dimensional data (Requirement B2). State-of-the-art techniques like HNSW [MY18] are constructed incrementally. Thus, online updates (Requirement B3.2) are well realizable. In general graph-based algorithms can not efficiently handle flexible target sets (Requirement B3.1). Focusing the search only on targets satisfying certain filter constraints is not possible, because thereby the connectivity of the graph is not ensured. To influence the accuracy of the search algorithm (Requirement B4), one can simply retrieve a higher number of candidates, e.g., raise the ef parameter in the HNSW search algorithm. However, in [YLFW20], the authors claim that this is less effective for the HNSW algorithm when compared to parameter tuning in quantization-based approaches.

[17] http://ann-benchmarks.com/ (Access: 04/09/2021)

Locality-Sensitive Hashing: LSH has been developed to overcome the curse of dimensionality. Accordingly, it can cope with a high-dimensionality (Requirement B2). The handling of flexible target sets (Requirement B3.1) constitutes a problem. If the number of buckets is high and the filter constraints are selective, the number of retrieved candidates can be 0. This problem can be solved by multi-probe LSH. However, selective filters still lead to an unnecessarily large amount of disk accesses. Online updates (Requirement B3.2) are possible. While the standard LSH algorithm does not provide methods to increase recall, this gets possible through multi-probe LSH. In this way, different demands of accuracy and run-time desired in Requirement B4 can be met.

Vector Quantization: Constructing an inverted index using vector quantization is a relatively simple solution for ANN on high-dimensional data. Thus it is widely applied [SZ03, BL12, BBS17] even though it does not provide state-of-the-art performance. Indexes based on a combination of inverted indexing and approximated distance calculation via product quantization achieve high performance. An index of product quantization sequences is appropriate to support flexible target sets (Requirement B2) as long as the preprocessing effort does not exceed the effort of the brute-force method. It allows retrieving only product quantization sequences satisfying the filter criteria for the approximated distance calculation leading to a candidate set of vectors that complies with the constraints. An inverted index based on quantization might be inefficient for small target sets since buckets might not contain vectors that comply with the filter criteria. Online updates (Requirement B3.2) can be implemented for both kinds of quantization-based indexing methods. To influence the accuracy, a certain number of low approximated distances can be post verified by exact distance calculation, and the number of partitions retrieved from an inverted index can be adjusted. Thus, the desired flexibility stated in Requirement B4 can be met.

4.4 RELATED WORK ON KNN SEARCH IN DATABASE SYSTEMS

There is already limited work done in integrating kNN operations in database systems. PostgreSQL, for instance, can be extended by PostGIS[18], which allows running kNN queries for low-dimensional (geographical) data. Index structures can be created with GiST (Generalized Search Trees) to speed up such operations. [BBS17] integrate vector similarity search for high-dimensional data into Spark for word embeddings. The authors state that simple index structures based on LSH [Cha02] or spherical k-means [DM01] are used to partition vectors for filtering. However, this might be only useful for a limited set of query types (see Section 4.3). A system called ADAMpro [GAKS14] adds ANN search techniques on top of a database system for multimedia retrieval. The vector database *Milvus* [WYG+21] provides a similarity search engine for high-dimensional data. In [WWW+20], the authors propose *AnalyticDB-V*, an execution engine for SQL queries that can perform ANN search on feature vectors filtered by additionally provided filter predicates.

There are two approaches to integrate approximated kNN-Joins into relational database systems: In [YLK10] approximated kNN-Joins based on z-order curves [Mor66] are integrated into relational database systems, which however, is only applicable for low-dimensional data. Just recently [YLFW20] investigated how common algorithms for approximated nearest neighbor search algorithms can be integrated into PostgreSQL leading to the development of the so called *PASE* extension.

[18] postgis.net (Access: 04/15/21)

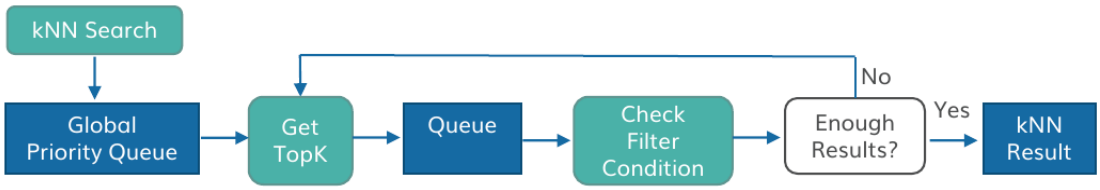


Figure 4.10: Execution of Queries with Filter Conditions with PASE Indexes

ANN-Joins based on Z-Curves In [YLK10], the authors aim at implementing an algorithm for kNN-Joins only based on SQL operations. In this way, the algorithm can be executed on any relational database system without integrating novel index structures and operations into the database system. Their implementation is based on z-order curve representations of multi-dimensional data. Thereby, the binary representations of each dimension[19] of a data point are interleaved into a single binary sequence. This leads to one-dimensional representations where two points with low distances in the multi-dimensional space tend to obtain representations with low distances. This allows the system to efficiently retrieve candidate solutions for the kNN problem by performing a range query on the z-value representation around the z-value of each query point. Then, the kNN-solution is obtained by an exact distance calculation. To increase the accuracy, the algorithm performs this process on several z-value representations which are generated from translations of all multi-dimensional points by random translation vectors $\mathbf{v_1}, \ldots, \mathbf{v_n}$. One major limitation of this approach is that it is only applicable to relatively low-dimensional data. The authors state that the algorithm is inappropriate for data with more than 30 dimensions. Accordingly, this approach is unsuitable for our requirements (see Requirement B2 in Section 3.5)

PASE In [YLFW20], the authors propose an extension for PostgreSQL called *PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension (PASE)*. While in our research, we focus on developing novel algorithms to handle ANN-Joins in a database, this work rather focuses on integrating existing techniques for ANN search operations deep into a relational database system. Therefore the authors especially focus on the storage representation of index structures and the implementation of the PostgreSQL interfaces to access the index data. They implement index structures for two ANN algorithms: *(1) IVFFlat*[20], a simple inverted index similar to the IVFADC index [JDS11] without product quantization, and *(2) HNSW* [MY18]. The authors claim that IVFFlat has a lower index building time and produces less storage overhead, while HNSW requires lower run-time, especially when a lower accuracy of the results is required. To handle queries with a target set that is restricted by filter constraints and therefore constitutes only a subset of the vectors in the index, the authors propose a specific execution schema shown in Figure 4.10. Hereby, the kNN operation continuously retrieves nearest neighbors and check the filter condition in an alternating fashion until k valid elements are retrieved. There is no open-source implementation available. The extension is available in the ApsaraDB for RDS online database service[21].

Regarding the requirements stated in Section 3.5, PASE provides efficient index structures for ANN search (Requirement B1), is able to handle high-dimensional data (Requirement B2), and can cope with different demands on precision and response time (Requirement B4). Furthermore, online updates are supported (Requirement B3.2). However, the

[19] We assume that all values are integers. The handling of floating-point values is discussed in [YLK10].

[20] https://github.com/facebookresearch/faiss/wiki/Faiss-indexes (Access 03/29/21)

[21] https://www.alibabacloud.com/help/doc-detail/147837.htm?spm=a2c63.p38356.a1.1.4fd41470hUUf8S (Access 03/29/21)

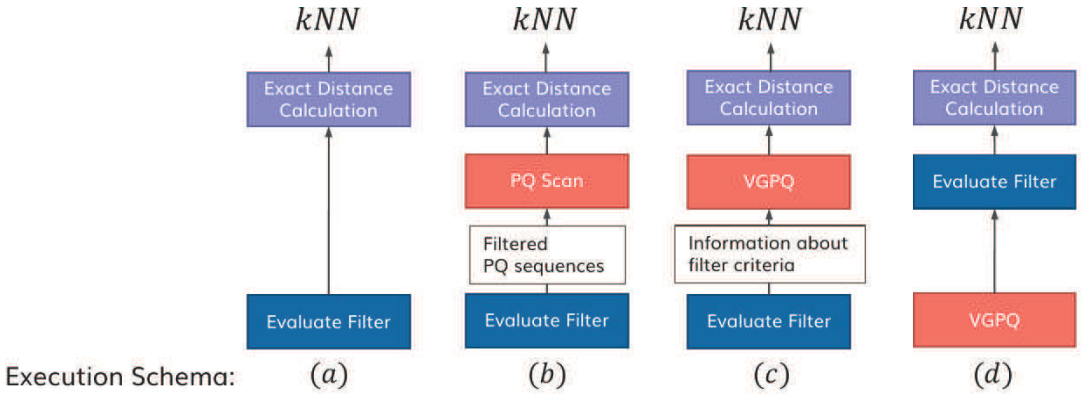


Figure 4.11: Execution of Queries with Filter Conditions with AnalyticDB-V

index structure is inefficient for processing kNN operations with highly selective filter constraints on the target set since it always performs a kNN search on the whole set of vectors. Therefore, it does not provide the required adaptivity according to (Requirement B3.1). Moreover, the index structures implemented by the extension rather focus on original kNN operations and do not provide specific optimizations for kNN-Joins.

AnalyticDB-V In a recent publication [WWW+20], the authors propose an analytical engine for the OLAP database system AnalyticDB [ZSW+19] to support so-called *hybrid queries* which involve ANN search operations and filter criteria. The database is proposed to store structured data as well as feature vectors generated for unstructured data objects, e.g., image descriptors. While this work rather focuses on vector representations for media data objects, the authors use similar indexing techniques as we proposed to use for word embeddings before in [Gün18, GTL19a]. In particular, they use an index, which constructs product quantization sequences for approximated distance calculation. Additionally, a novel VGPQ index combines quantization-based inverted indexing with the product quantization sequences. To solve queries with filter criteria, they consider four execution schemes shown in Figure 4.11. The brute-force method (a) uses an index only for the filter constraint on the structured. Accordingly, exact distances are calculated for all vectors complying with the filter criteria to solve the query. In (b), only the product quantization sequences are used without inverted indexing. This allows the algorithm to apply the filter criteria before the execution of the ANN search. (c) resembles the schema we also used for our execution of kNN-Joins in [GTL19a] and also the execution schema of PASE (see Figure 4.10). Here, the filter criteria are also applied before the execution of the ANN search. To use the inverted index, the information, which vectors comply with the filter, is provided as an additional parameter set to the index operation. This allows the inverted index to only retrieve vectors that comply with the filter. In contrast to the schema of PASE, an exact distance calculate is only done for valid candidates leading to a more efficient search. However, in the case of a selective filter, a large set of buckets has to be retrieved from the inverted index. In (d), the ANN search is executed before the filtering, which only makes sense for filter criteria with low selectivity. Based on a cost model applied to the query, one of the four schemas is used.

AnalyticDB-V provides an efficient solution for ANN search for high-dimensional data (Requirement B1 and B2 in Section 3.5), and it allows to adjust to different demands on precision and response time (Requirement B4). It also allows online updates (Requirement B3.2). However, it only focuses on the execution of kNN search operations and does not specifically consider kNN-Joins. While it is a good idea to provide different execution schemes for different filter criteria, the execution schemes do not consider the execution of multiple queries for the same set of targets. Especially in schema (c), the filter criteria have to be checked during the retrieval of candidates from the inverted index for each query vector separately. In this regard, the handling of flexible target sets (Requirement B3.1) could be improved.

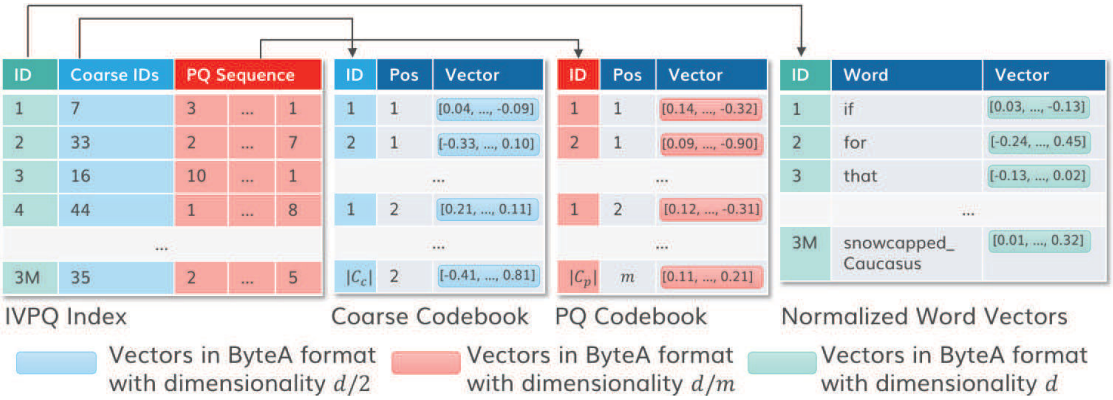


Figure 4.12: Index Data Structure

4.5 ANN-JOINS FOR RELATIONAL DATABASE SYSTEMS

We decided to use a quantization-based approach for the implementation of the ANN-Join algorithm. We made this decision based on our review of common ANN techniques in Section 4.3 with regard to the requirements for ANN-Joins on word embeddings stated in Section 3.5. However, it should be noticed that for kNN-Join operations with small query and target sets, an exact algorithm is applicable using the brute-force method to calculate all distance values between pairs of query and target vectors. Moreover, for relatively small query and target sets, the product quantization distance calculation described in Section 4.2.4 is applicable. Therefore, an index of product quantization sequences needs to be maintained. In all other cases, a combination of inverted indexing and product quantization distance calculation should be used, which is also used by several other ANN search engines [JDS11, WWW+20, GSL+20].

While previous works on ANN search in high-dimensional spaces proposed algorithms for single ANN queries, we developed an adaptive algorithm for ANN-Joins processing multiple kNN-queries at a time to satisfy Requirement B1 in Section 3.5. We describe the index structure for our algorithm in Section 4.5.1 and the search algorithm itself in Section 4.5.2. To increase the performance for queries with a restricted target set T requested in Requirement B3.1, we identified two opportunities for optimizations outlined in Section 4.5.4. Those are addressed by the *candidate number estimator* explained in Section 4.5.5 and the *flexible product quantization* proposed in Section 4.5.6. Additional minor optimizations are described in Section 4.5.7. Afterward, we discuss the influence of several search parameters on precision and run-time and give advice for tuning them in Section 4.5.8. For supporting fast kNN-Joins on word2bit vectors (see Section 2.2.7), we provide specific optimizations explained in Section 4.5.9. A comprehensive evaluation of our ANN-Join approach is done in Section 4.6.

4.5.1 Index Architecture

We propose to use a combination of the inverted multi-index [BL12] described in Section 4.2.4 and an index of product quantization sequences for approximated distance calculation. The data structure of our index is shown in Figure 4.12. For each d-dimensional vector, an index entry is created in the *IVPQ Index* table. Every entry contains an id to reference it, equivalent to the id in the vector tables (see Section 4.1.2). For the inverted multi-index, we divide the vectors into two partitions so that each entry consists of PQ

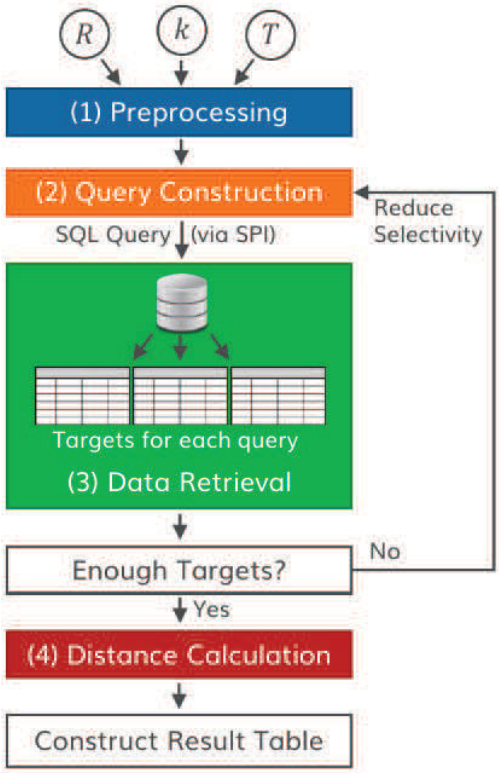


Figure 4.13: ANN-Join Algorithm

Algorithm 4.2: kNN-JOIN(R, k, T, α)

Input: query vector set R, results size k, target vector set T, amplification factor α
Output: k nearest neighbors for each query vector

1 $\mathcal{D}_{pre} =$ PREPROCESSING(R, T, α, k);
2 $R' = R$;
3 $j = 1$;
4 **while** $R' \neq \emptyset$ **do**
5 **for** $\mathbf{r_i} \in R$ **do**
6 $C^* =$ COARSEQUANTIZE$(\mathbf{r_i})$;
7 $centr(i) =$ SELECTPAR(C^*, T, α, k, j);
8 **end**
9 $query =$ CONSTRUCTQUERY$(centr, T)$;
10 $T_{sub} =$ EXECUTE$(query)$;
11 $R' = \{\mathbf{r_i} \mid |T_{sub}(i)| < \alpha \cdot k\}$;
12 $j = 2 \cdot j$;
13 **end**
14 **for** $\mathbf{r_i} \in R$ **do**
15 **for** $\mathbf{t} \in T_{sub}(i)$ **do**
16 $d =$ DISTFUNC$(\mathbf{r_i}, t, \mathcal{D}_{pre})$;
17 UPDATE$(top_k[\mathbf{r_i}], d)$;
18 **end**
19 **end**
20 **return** top_k

Figure 4.14: Pseudo-Code

sequences of length two. To perform an inverted search, the combination of the two *coarse ids* refers to a partition to which the vector belongs. Those ids c_1 and c_2 are represented by a single id $id_c = c_1 \cdot |C_c| + c_2$ in the *Coarse IDs* column. Each of the two coarse ids corresponds to a $\frac{d}{2}$-dimensional centroid[22]. Those centroids are stored in the *Coarse Codebook* table, where the Pos value refers to the partition (either 1 or 2). In addition, a PQ sequence of length m allows performing approximated distance calculations. The centroids of subvectors for the product quantization with a dimensionality of $\frac{d}{m}$ are stored in the *PQ Codebook* table. Each of those PQ centroids has an id that corresponds to codes in the product quantization sequences and a position $Pos \in \{1, \ldots m\}$ corresponding to the position of the subvector it is calculated for $(u_1, \ldots, u_m)$. The normalized vectors can be used for the distance calculation of candidate vectors explained in Section 4.5.3.

4.5.2 Search Algorithm

Figure 4.13 shows a flow chart, and Figure 4.14 the pseudo-code of our algorithm. As input parameters, the algorithm gets a set of query vectors $R = \mathbf{r_1}, \ldots \mathbf{r_n}$, a set of target vectors T, represented as a set of index entry ids, and the desired number k of nearest neighbors. Furthermore, the configuration parameter α determines the minimum number of targets per neighbor that has to be considered for the search process. A higher value of α leads to a higher precision of the result set.

The algorithm consists of four steps:
At first, there is a *preprocessing step* (Line 1), which is necessary for the product quantiza-

[22] If d is an odd number, one centroid set should be created with a dimensionality one greater than the other set.

tion-based distance calculation described in Section 4.2.4. As a result, pre-computed distance values of subvectors are obtained and stored in $\mathcal{D}_{pre}$. Details about the preprocessing are provided in Section 4.5.6.
After that, data is retrieved from the index in multiple iterations. Therefore, a set of query vectors to consider R' is initialized with the set of all queries R, and the iteration count j is set to 1. In the *query construction step* (Line 4 to 9), the retrieval of database entries from the inverted index is prepared. This involves the calculation of the coarse quantization C^* for every query vector $\mathbf{r_i}$ in Line 6 with the algorithm described in [BL12]. It returns a sequence of the coarse centroid id tuples corresponding to the entries in the *Coarse IDs* columns in the IVPQ table (see Figure 4.12) in increasing order according to the distance between the coarse centroids and the respective partitions of the query vector $\mathbf{r_i}$. Then, a set of partitions to be retrieved from the index for $\mathbf{r_i}$ is determined using the SELECTPAR function and added to $centr(i)$. A naïve implementation of SELECTPAR would select the first j closest partitions for each query from C^*. In Section 4.5.5, we propose a more sophisticated approach that uses the candidate number estimator.
After that, a single SQL query is constructed to retrieve the data for all query vectors from the index using the SPI in the *data retrieval step* in Line 10. Then, the query vectors R' for which not enough index entries ($< \alpha \cdot k$) have been retrieved are determined. For those, another query construction and retrieval iteration is done where a higher number of closest partitions is selected by increasing j.
If the number of targets is sufficient, the distance values between every query vector $\mathbf{r_i}$ and its respective index entries $T_{sub(i)}$ are calculated by a distance function DISTFUNC (*distance calculation step*). We elaborate more on the distance function in Section 4.5.3. The best candidates for the kNN operation are held in a sorted list top_k which is updated after every distance calculation.

4.5.3 Distance Calculation

Besides the adaptive number of partitions and vectors to be considered by the kNN-Join algorithm influenced by α, the trade-off between precision and run-time also depends on the distance function, namely: *(1) exact calculation*, *(2) product quantization*, and *(3) product quantization with post verification*. Product quantization is the fastest one but also provides the lowest precision. It calculates distance values as described in Section 4.2.4. The exact calculation is too slow, especially for a large number of target vectors and a large α. Method (3) strikes a balance between both extremes and therefore represents the default distance function. In the first place, it calculates the approximated distance values using product quantization for $k \cdot \alpha$ targets. Second, it refines the $k \cdot pvf$ best candidates with the exact method to obtain the final top-k. Here, the post verification factor pvf is the primary factor to influence the precision of the kNN computation. By adjusting it, the user can control the trade-off between precision and run-time as desired in Requirement B4 of Section 3.5. In the evaluation in Section 4.6.2, we further investigate this.

4.5.4 Optimization Capabilities

We identified two promising opportunities for optimizing the search algorithm of Section 4.5.2: an estimator for the candidate set size to implement the SELECTPAR function in Line 7 in the algorithm in Figure 4.14 and flexible product quantization.

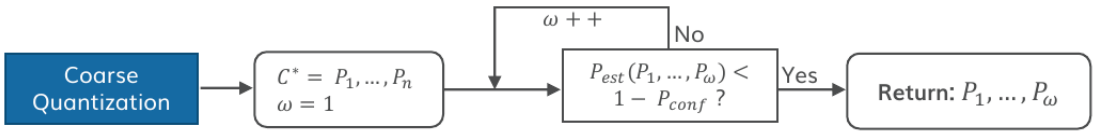


Figure 4.15: Confidence Estimation

Candidate Number Estimator: In Section 4.2.4, we described the advantages of inverted indexing for ANN. However, inverted indexing, in general, is poorly suitable when the target set T is only a subset of all vectors in the index T_I, i.e., $|T| \ll |T_I|$. In those cases, a larger number of partitions need to be considered to retrieve enough candidate vectors. However, it is not obvious how many partitions should be examined. AnalyticDB-V [WWW+20] solves this problem for a single ANN query by continuously retrieving partitions until enough candidates are identified present in T. However, in this way, multiple index accesses are necessary. The use of inverted multi-indexing enables fast lookups and therefore allows us to efficiently handle a large number of small partitions. This increases the accuracy, but it might lead to many index accesses until the desired number of candidates is reached. To solve this problem, we propose in Section 4.5.5 a method to estimate the number of targets in T observed when a certain number of partitions is read out from the index. Based on this, we can pick the optimal number of partitions (Figure 4.14 Line 7) and retrieve all necessary partitions for all query vectors with one or only a few index accesses.

Flexible Product Quantization: Usually, ANN search approaches are optimized to handle queries with large target sets because those search tasks produce the highest computational effort. In the case of the ANN-Join problem, tasks with a large query vector set also require a high computational effort. However, an index structure designed for queries with large target sets might not effectively reduce the run-time for those tasks. Especially for the product quantization distance calculation, the run-time required for the preprocessing step might exceed the execution time of the actual distance calculation for small target sets. Between the preprocessing effort and the effort for the approximated distance calculation for each candidate exists a trade-off. This trade-off depends on the configuration parameters set for the construction of the product quantization index. To design a solution to deal with target sets of different sizes without constructing multiple indexes, we introduced a method for flexible product quantization in Section 4.5.6. This allows us to construct a single product quantization index which enables the search algorithm to switch between a comprehensive preprocessing and a fast preprocessing during run-time depended on the target set size.

4.5.5 Estimation of the Number of Targets

In Line 7 in Figure 4.14, the SELECTPAR function should select a suitable set of partitions from the list of nearest partitions C^*. To achieve this goal, we estimate a suitable number $\omega \leq n$ of nearest partitions, where $n = |C^*|$ constitutes the number of all partitions. The maximal *coarse order* ω is selected in a way, that the probability P_{est} that it is necessary to run further database requests for the query vector $\mathbf{r_i}$ to obtain enough candidates (at least $k \cdot \alpha$) is lower than a certain value $1 - P_{conf}$. This is done by iteratively incrementing ω until the confidence value $1 - P_{est}$ obtained by a probabilistic model is higher than P_{conf}

(see Figure 4.15). The estimation relies on statistics about the distribution of the index. Those contain the relative sizes of all partitions $P_1, \ldots, P_n$ compared to the whole index size (the total number of vectors).

For the estimation, we consider the set of all index entries T_I, the target set of the current query T_i, and a set of selected partitions $T_p = P_1 \dot{\cup} \ldots \dot{\cup} P_\omega$. We estimate the probability $1 - P_{est}$ that $T_{sub}(i)$ contains at least $\beta = k \cdot \alpha$ candidates, which corresponds to the condition in the algorithm of Figure 4.14 in Line 11. The number of candidates $|T_{sub}(i)|$ corresponds to the cardinality of $T_i \cup T_p$. For the estimation, we leverage a hypergeometric probability distribution (Equation (4.3)) which describes the probability to get s successes by drawing M elements out of a set of N elements without replacement. In our case, s is the desired number of targets in $T_i \cup T_p$, M is the cardinality of T_p, and N equals $|T_I|$. The probability $1 - P_{est}$ of drawing at least β target vectors from the set T_I of all vectors in the index can be calculated with Equation (4.4) by using the cumulative distribution function.

$$h(X = s \; ; \; |T_I|, |T_i|, |T_p|) = \frac{\binom{|T_p|}{s}\binom{|T_I|-|T_p|}{|T_i|-s}}{\binom{|T_I|}{|T_i|}} \tag{4.3}$$

$$\mu = |T_i| \cdot \frac{|T_p|}{|T_I|} \quad \sigma^2 = |T_i| \cdot \frac{|T_p|}{|T_I|} \cdot \Big(1 - \frac{|T_p|}{|T_I|}\Big) \cdot \frac{|T_I|-|T_i|}{|T_I|-1}$$

$$1 - P_{est} = h_{cdf}(\beta - 1 \; ; \; |T_I|, |T_i|, |T_p|) = 1 - \sum_{s=0}^{\beta-1} \frac{\binom{|T_p|}{s}\binom{|T_I|-|T_p|}{|T_i|-s}}{\binom{|T_I|}{|T_i|}} \tag{4.4}$$

However, because of the complexity of the computation of the binomial coefficients, we have to use an approximation based on the normal distribution stated in Equation (4.5). To obtain the approximation, we use the mean μ and the variance σ^2 from the hypergeometric distribution (see Equation (4.3)). Here, the number x is a *specific* number of targets which corresponds to $|T_i \cup T_p|$.

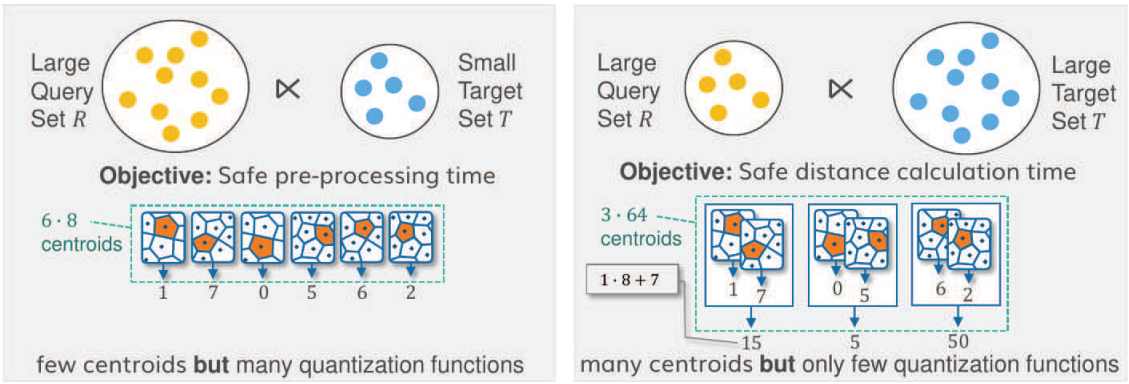
$$N(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\Big[- \frac{(x-\mu)^2}{2\sigma^2} \Big] \tag{4.5}$$

The approximation of the probability of getting at least $\beta - 1$ targets (Equation (4.6)) is then obtained by its cumulative distribution function. The addition of 0.5 serves as a continuity error correction. It is added to the formula since the hypergeometric distribution is a discrete probability distribution. Since β is an integer value, $x < \beta - 1$ corresponds to $x < \beta - 0.5$.

$$h_{cdf}(\beta - 1; \mu, \sigma^2) \approx 1 - \sum_{s=0}^{\beta-1} N(s; \mu, \sigma^2) \approx 1 - \frac{1}{2} \cdot \left(1 + erf\Big(\frac{(\beta-1) + 0.5 - \mu}{\sqrt{2}\sigma}\Big)\right) \tag{4.6}$$

The probability h_{cdf} of getting enough targets can be increased by raising the number of partitions in T_p, which corresponds to the coarse order ω. The algorithm in Figure 4.15 chooses ω so that it is minimal and h_{cdf} is higher than a specific probability P_{conf}, also termed the *confidence value*. From experimental results, we noticed that 0.8 seems to be a suitable value to achieve a high response time for the algorithm.

The implementation of the SELECTPAR function (Line 7 in Figure 4.14) estimates the maximal corase order ω with the proposed algorithm (see Figure 4.15) depended on α, k, $T_i = T$, and the previously determined ordered list of partition ids C^*. If not enough



(a) Long Code Mode (b) Short Code Mode

Figure 4.16: PQ Sequences of Flexible Product Quantization

candidates could be retrieved in the data retrieval step (3) for specific query vectors, SELECTPAR obtains a larger target set by using a modified $\alpha' = \alpha \cdot j$ for the estimation. By increasing α, by multiplication with the iteration count j, a more conservative estimation is done, and the selectivity in the data retrieval step is reduced. However, the run-time of the subsequent database queries is presumably much lower than the run-time of the first query since partitions are only retrieved for a small number of query vectors, where previous retrieval steps do not deliver enough candidates.

4.5.6 Flexible Product Quantization

The product quantization index provides two parameters: the number of subvectors m and the number of centroids per quantizer $|C|$. The optimal setting of both parameters depends on the desired precision and the response time, as well as on the typical number of distance calculations $\alpha \cdot k$, which are performed for every query vector. In general, higher values of m and $|C|$ correspond to higher precision and higher response times.

If the target set size $\alpha \cdot k$ is large, the computation of the distances (Line 16 in Figure 4.14) is the most time-consuming step, whereas, for small target sets, the computation time of the preprocessing step (Line 1) becomes more and more prevalent. Since a low value for m, i.e., a low number of subvectors, corresponds to a faster distance calculation, the product quantization speed for large target sets depends mainly on m. However, with a decreasing number of vectors $\alpha \cdot k$, the proportion of the computational effort for the preprocessing step (Line 1), which is mainly influenced by $|C|$, increases. By decreasing the number $|C|$, the search process gets faster. To be efficient in both situations, we introduce a flexible product quantization search procedure.

For product quantization search with a small number of distance calculations $\alpha \cdot k < Th_{flex}$, an index is created with a large number of subvectors $m = 2 \cdot m'$, but only a few centroids in C. Figure 4.16a visualize this case, which we call the *Long Codes Mode* since the pq sequences consist of a larger number m of ids. Long codes are specifically useful for small target sets, where fewer candidates per query need to be retrieved from the inverted index. For a larger number of distance calculations, the number of distances to sum up for each distance calculation (see Equation (4.2)) can be reduced by

pre-calculating squared distances for pairs of centroids $\langle \mathbf{c_j}, \mathbf{c_{j+1}} \rangle$ and pairs of subvectors $\langle u_j(\mathbf{r}), u_{j+1}(\mathbf{r}) \rangle$:

$$d(\langle u_j(\mathbf{r}), u_{j+1}(\mathbf{r}) \rangle, \langle \mathbf{c_j}, \mathbf{c_{j+1}} \rangle)^2 = d(u_j(\mathbf{r}), q_j(u_j(\mathbf{y})))^2 + d(u_j(\mathbf{r}), q_{j+1}(u_{j+1}(\mathbf{y})))^2 \quad (4.7)$$
$$where: \ \mathbf{c_j} = q_j(u_j(\mathbf{y})), \mathbf{c_{j+1}} = q_{j+1}(u_{j+1}(\mathbf{y})), j \in \{2 \cdot i | i \in \mathbb{N}\}$$

The distance calculation can then be expressed by the following equation:

$$\hat{d}(\mathbf{r}, \mathbf{y})^2 = \sum_{j=1}^{m'} d(\langle u_{2j-1}(\mathbf{r}), u_{2j}(\mathbf{r}) \rangle, \langle \mathbf{c_{2j-1}}, \mathbf{c_{2j}} \rangle)^2 \quad (4.8)$$

To efficiently calculate this, the product quantization sequences consisting of m numbers can be transformed into sequences of $m' = \frac{m}{2}$ numbers. Therefore, all centroid id pairs $\langle id(\mathbf{c_j}), id(\mathbf{c_{j+1}}) \rangle$ can be transformed into single ids:

$$id(\mathbf{c_j}, \mathbf{c_{j+1}}) = id(\mathbf{c_j}) \cdot |C| + id(\mathbf{c_{j+1}}) \quad (4.9)$$

This is called the *Short Codes Mode* displayed in Figure 4.16b. Optimal settings for the threshold Th_{flex} are discussed in Section 4.6.4.

4.5.7 Further Optimizations

Target List for Product Quantization Search In Figure 4.14 in Line 10, the algorithm reads out candidates from the inverted index in the form of product quantization sequences. A naïve algorithm would directly iterate through the pq sequences and perform the product quantization distance calculation (target-wise) for all query vectors where one of the nearest centroids belongs to the partition of the pq sequence. However, to execute product quantization efficiently, the precomputed distances should stay in the cache. Since the precomputed distances are specific for the query, it is necessary to collect all product quantization sequences and assign them to the query vectors in the first place. Afterward, the distance computation can be done query-wise. So, all precomputed distances specific for a query can stay in the cache. Moreover, the approach of [AKLS15] could be used to further improve memory locality to speed up the product quantization search. Thereby, product quantization sequences are compressed to fit into SIMD cache lines.

Prefetching As displayed in Figure 4.13, we collect the targets in Line 10 of the search algorithm (Figure 4.14) and assign them to target lists for the query vectors they should be compared to. This requires a lot of random memory access to the lists of targets. To speed up this step, we prefetch the target list entries, which have to be updated next from time to time. We tested the effect of the prefetching with our algorithm on a query with 10,000 queries, 100,000 targets (300-dimensional vectors), $\alpha = 100$, and $k = 10$. For this query, the construction time of the target list could be reduced by $\approx 35\%$, from 1.4 seconds to 0.9 seconds. For more details about that, one can take a look at the code[23].

[23] https://github.com/guenthermi/postgres-word2vec/blob/master/freddy_extension/ivpq_search_in.c (Access 05/04/21)

Name	Symbol	Domain	Description
Post Verification Factor	pvf	$\mathbb{N}^+, pvf \leq \alpha$	number of exact distance calculation per result
Amplification Factor	α	$\mathbb{N}^+, \alpha \geq pvf$	minimal number of candidates per result
Flexible PQ Threshold	Th_{flex}	$\mathbb{N}$	Threshold for switching between pq distance calculation with long codes and short codes
Confidence Probability	P_{conf}	$\mathbb{R}, P_{conf} \in [0, 1]$	minimal confidence of retrieving at least $k \cdot \alpha$ candidates

Table 4.1: Search Configuration Parameters

Fast Top-K Update In Line 17 of the algorithm in Figure 4.14, the top_k gets updated after every distance calculation. If the distance value is lower than every other index entry, the new index entry has to be inserted into this array of current nearest neighbors. However, this can be time-consuming since every other array element with a larger distance has to be moved. For a large top_k, the updates can be accelerated by first adding new candidates in a buffer. If this buffer gets full or all distance calculations are done, all candidates are added to the top_k in one run. This is particularly useful if post verification (see Section 4.5.3) should be done, and thus a large set of candidates is required in the first place. Alternatively, one can use a linked list instead of an array for the top_k. However, linked lists are space-consuming which could become a problem for large query sets.

4.5.8 Parameter Tuning

Table 4.1 gives an overview of the parameters to configure the execution of the kNN-Join operation. By adjusting pvf and α, the user can influence the trade-off between precision and run-time as required by Requirement B4 in Section 3.5. Section 4.6.2 investigates the influence on precision and run-time for both parameters. Th_{flex} and P_{conf} influence primarily the run-time. Their optimal values for discussed in Section 4.6.3.

4.5.9 kNN-Joins for Word2Bits

Binary Word embeddings trained with the word2bit method [Lam18] described in Section 2.2.7 can be used to very efficiently perform kNN search operations. Thereby, the vectors are stored as bit vectors where 1 refers to $\frac{1}{3}$ and 0 to $-\frac{1}{3}$. Those vectors are space efficiently stored in vector tables as described in Section 4.1.2.

Cosine Similarity Calculation: Since binary word2bit vectors have the same magnitude of $\frac{1}{3}$ in each dimension, all vectors with dimensionality n have the same length of $\frac{1}{3}\sqrt{n}$. Therefore, the cosine similarity of two vectors $\mathbf{u}$ and $\mathbf{v}$ with length n is proportional to the scalar product of the two vectors:

$$sim_{cos}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}| \cdot |\mathbf{v}|} = \left(\frac{3}{\sqrt{n}}\right)^2 (\mathbf{u} \cdot \mathbf{v}) = \frac{9}{n}(\mathbf{u} \cdot \mathbf{v}) \tag{4.10}$$

According to Equation (4.10), the similarity between two vectors is proportional to the count of dimensions with equal binary values substracted by the number of dimensions

with unequal binary dimensions. Formally, this relation between the vectors and their binary representations $b(\mathbf{u})$ and $b(\mathbf{v})$ is defined in Equation (4.11):

$$sim_{cos}(\mathbf{u},\mathbf{v}) \propto \mathbf{u}\cdot\mathbf{v} = \sum_{i=1}^{n}\frac{1}{9}\Big[2\Big(b(\mathbf{u})_i \cdot b(\mathbf{v})_i\Big) - 1\Big] \propto \underbrace{\Big(\sum_{i=1}^{n} b(\mathbf{u})_i \wedge b(\mathbf{v})_i\Big)}_{n-d_{ham}(b(\mathbf{u}),b(\mathbf{v}))} - \frac{n}{2} \tag{4.11}$$

Consequently, the cosine distance of binary word2bit vectors is linearly dependent on the Hamming distance between the binary representations. Based on this observation, we can solve the kNN problem with respect to the Euclidean distance (or the cosine distance) for those vectors by solving the kNN problem with respect to the Hamming distance for their binary representations.

$$kNN(\mathbf{r},T) = \underset{\{\mathbf{t_1},\ldots,\mathbf{t_k}\}\in T^{[k]}}{\arg\min} \sum_{i=1}^{k} d(\mathbf{r},\mathbf{t_i}) = \underset{\{\mathbf{t_1},\ldots,\mathbf{t_k}\}\in T^{[k]}}{\arg\min} \sum_{i=1}^{k} d_{ham}(b(\mathbf{r}),b(\mathbf{t_i})) \tag{4.12}$$

Implementation: To efficiently perform this calculation, we can utilize bit-wise instructions, which are available on modern standard CPUs. Specifically, we use a combination of the XOR and the population count (popcnt) operation, which calculates the Hamming weight[24] of a bit-vector, to efficiently determine the Hamming distance:

$$d_{ham}(b(\mathbf{r}),b(\mathbf{t_i})) = popcnt\Big(XOR\Big(b(\mathbf{r}),b(\mathbf{t_i})\Big)\Big) \tag{4.13}$$

Both operations, XOR and $popcnt$, can be executed by one CPU instruction on sequences of 64 bit of the two bit-vectors at a time. The sum of this result constitutes the Hamming distance. To implement a kNN-Join operation for word2bit vectors, we created a UDF which executes kNN queries according to Equation (4.13) in a batch-wise manner. Our evaluation in Section 4.6.6 shows that this method already achieves sufficient performance to process queries on the available pre-trained binary word2bit models. To further improve the kNN search on very large binary vector sets, recently, several indexing techniques have been proposed [EAT19, OB16] to avoid linear scans.

A limitation of this approach is that the query vectors need to be bit-vectors. This constitutes a problem if vectors need to be generated for text values, which are not present in the vocabulary of the word2bits model. The averaging method [ALM17] explained in Section 2.2.5 to index new terms leads to vectors with values not in $\{\frac{1}{3},-\frac{1}{3}\}$. As a workaround, one could quantize each value of the resulting vector to one of the two values $\frac{1}{3}$ or $-\frac{1}{3}$ to obtain valid word2bit vectors.

4.6 EVALUATION

In this section, we first evaluate our adaptive kNN-Join implementation for varying post verification factors and α values and compare them to the basic batch-wise product quantization approach (see Section 4.6.2). Moreover, we provide a detailed run-time investigation for the sub-routines of the kNN-Join given different query and target set sizes (Section 4.6.3). To examine the influence of the flexible product quantization, we evaluate the impact of using the short and long codes model on the precomputation and distance calculation in Section 4.6.4. In Section 4.6.5, we evaluate the accuracy of the target size estimator which is explained in Section 4.5.5. Finally, we evaluate the kNN-Join algorithm for word2bits vectors in Section 4.6.6.

[24]The Hamming weight is defined as the number of 1s in a bit-vector.

	Google News (GN)	Twitter (TW)
Size	$3,000,000$	$1,193,514$
Dimensionality	300	100
Coarse Centroids	$2 \cdot 32$	$2 \cdot 20$
Product Quantization	$m = 30,\ \|C\|^* = 32$	$m = 10,\ \|C\|^* = 32$
Confidence	$P_{conf} = 0.8$	$P_{conf} = 0.8$
Threshold (for flexible product quantization)	$Th_{flex} = 15,000$	$Th_{flex} = 15,000$

$*$ number of centroids for each quantizer

Table 4.2: Dataset and Index Characteristics

4.6.1 Experimental Setup

We use two different datasets of word embeddings to evaluate our approach, the popular Google News word2vec dataset (W2V-GN in Table 3.1) and a dataset of 100-dimensional word vectors trained on data from Twitter[25] with GloVe [PSM14]. We use Python scripts to create the IVPQ index structures (see Figure 4.12) on these datasets for our adaptive ANN-Join algorithm with the parameters shown in Table 4.2. In addition, we create for the baseline approach a pure product quantization index with the same parameters used to generate the product quantization sequences in the IVPQ index table. We integrated our implementation of the ANN-Join algorithm into the word embedding database system FREDDY as a user-defined function.

As a baseline, we use the exhaustive product quantization search as described in [JDS11], which can easily be generalized to an ANN-Join operation. Basically, it makes no use of inverted indexing and thus calculates approximated distance values between any query vector in R and any target vector in T to determine the kNN results. To make the comparison fair, we implemented a batch-wise search algorithm as UDF, like it is done for the adaptive search algorithm.

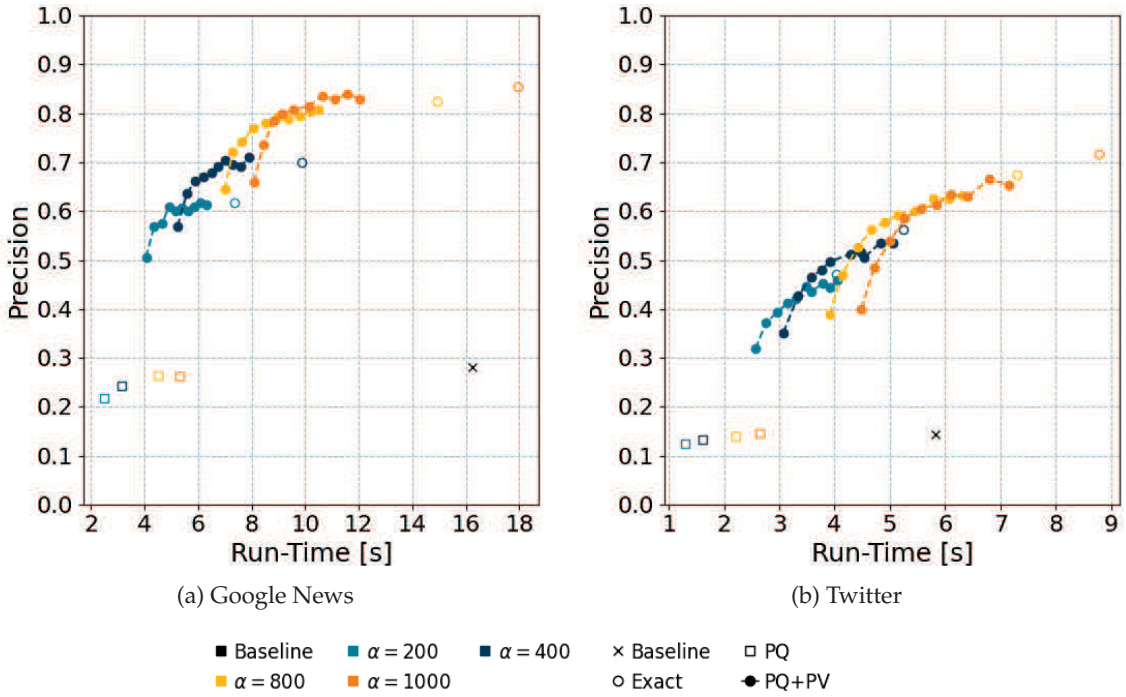
We guarantee repeatability by publishing the implementation in the Github repository of FREDDY[26]. The machine we run the evaluation on is a Lenovo ThinkPad 480s with 24GB main memory, an Intel i5-8250U CPU (1.6GHz), and a 512GB SSD. The computation runs only on a single core in a PostgreSQL instance on a Ubuntu 18.04 Linux System.

4.6.2 Influence of Index Parameters on Precision and Execution Time

In Figure 4.17, the execution time and precision curves for different α values and increasing pvf values are shown. All the kNN-Joins are executed on 5,000 query and 100,000 target vectors with $k = 5$. The post verification factors used for the computation are $10, 20, \ldots, 100$. The precision is determined by calculating the amount of nearest neighbor results of a query vector which concur with the exact results relative to the number of k. Since doing the exact calculation for all query vectors of a kNN-Join is very time-consuming, we draw bootstrap samples of the query vectors of size 100 to derive an estimation of the actual precision value by determining the precision of the results of the samples. The measurements for every configuration are done 20 times and the median run-time values and the mean precision values are determined. The value of $\alpha \cdot k$ is always lower than Th_{flex}. Thus, the *Long Codes Mode* is used.

[25] https://nlp.stanford.edu/projects/glove/ (Access: 06/10/21)

[26] ANN-Join implementation: https://github.com/guenthermi/postgres-word2vec/blob/master/freddy_extension/ivpq_search_in.c (Access: 06/10/21)



(a) Google News (b) Twitter

Figure 4.17: Evaluation of Execution Time and Precision

As one can see, for most of the chosen values of pvf and α, the adaptive search with PQ distance calculation has the shortest execution time and also outperforms the product quantization baseline method in terms of run-time by achieving similar precision values. Join operations with exact distance calculation have significantly longer execution times than the other methods, however achieving the highest precision value. Nevertheless, post verification might be the better choice in most cases since it achieves high precision values while being much faster than the exact computation. For increasing values of pvf, the execution time, but also the precision generally increases. Regarding the α values, one can also observe that higher values lead to higher precision values at the expense of execution time.

The post verification method is significantly slower than the product quantization distance calculation method, even though pvf has a low value. This is the case since the use of post verification requires calculating at least k exact distance values for each query vector. Furthermore, it needs to retrieve the raw vector data for every target vector, which has to be considered for distance calculation. Moreover, it is necessary to hold these vectors in memory until the distance computation starts. During the distance computation, the vectors of the currently best candidates have to be stored together with the product quantization sequences in a separate TopK list to apply the post verification step later. For high values of pvf, on the one hand, the post verification step gets time-consuming, while on the other hand, more updates of the TopK lists are required during the distance calculation step.

4.6.3 Performance of Subroutines

We evaluate the performance of the search algorithm by measuring the execution time of certain subroutines of the algorithm denoted by numbers 1 to 4 in Figure 4.13. This is done for different cardinalities of query sets R and target sets T. The query and target

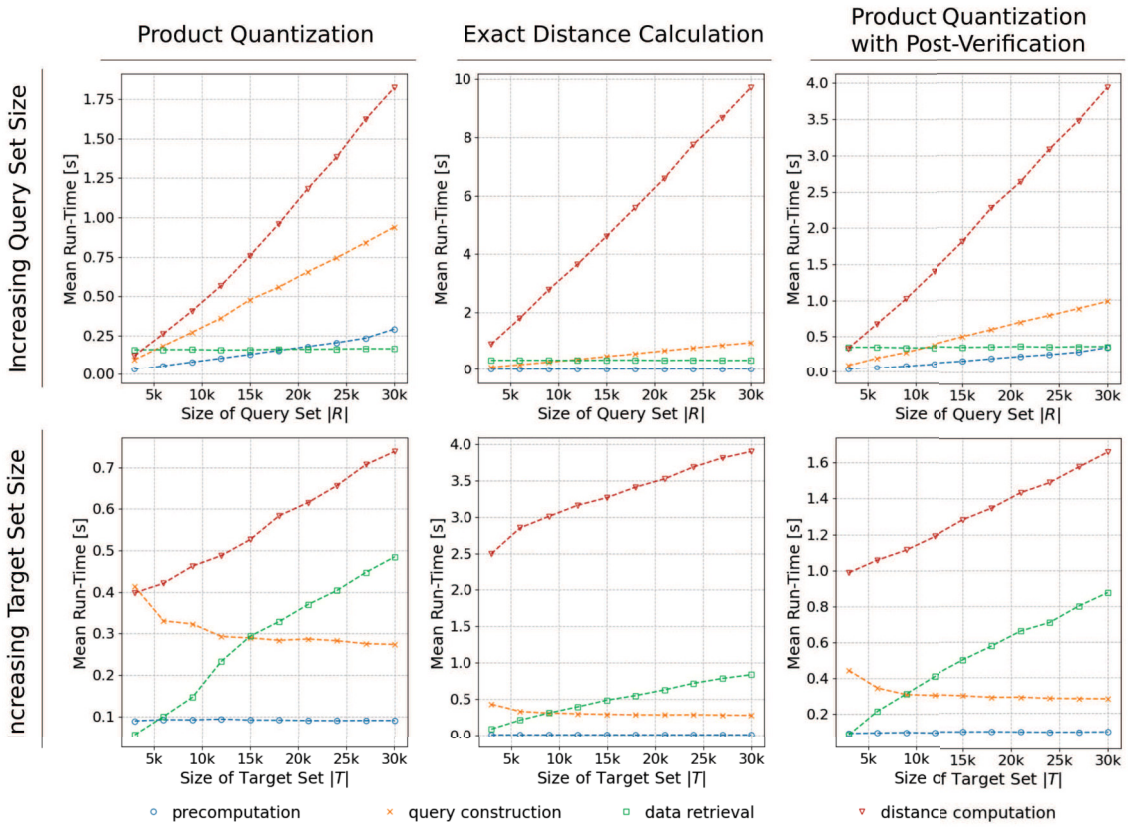
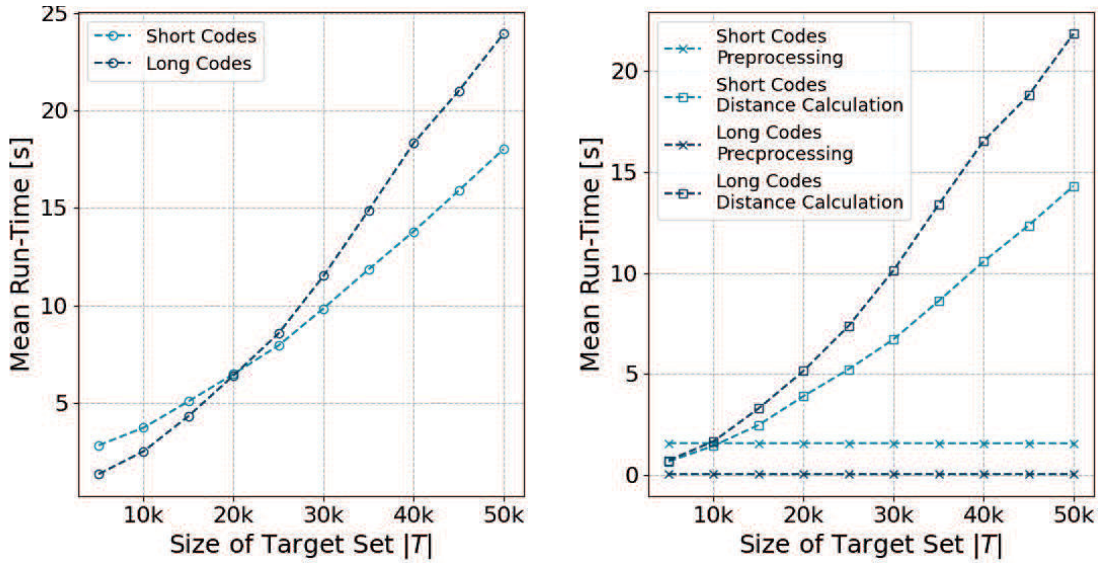


Figure 4.18: Time Measurement for increasing sizes of query set R and target set T

vectors are sampled from the whole set of word embeddings of the Google News dataset. The results of our measurements are shown in Figure 4.18 for different values of $|R|$ and $|T|$. For the measurements, we set $\alpha = 100$ and $pvf = 10$. We use a fixed target set size of $10,000$ while increasing $|R|$ and a query set size of $10,000$ while increasing $|T|$. All measurements are done five times and the average value is determined. The distance computation time increases with the query set size as well as with the number of target vectors. The query construction time only increases with an increasing query set size. If the query set size is fixed, the query construction time slightly decreases with increasing target set size because a higher number of partitions has to be determined for every query vector in case the number of targets is very low. The main effort during the query construction is the calculation of the coarse quantization for every query vector. Since this process does not change with the number of target vectors, the execution time is rather constant. The data retrieval time effort is nearly constant for an increasing number of query vectors, while its execution time increases if the target vector set grows. The preprocessing has to be done per query vector. Therefore only the query set size influences its execution time.

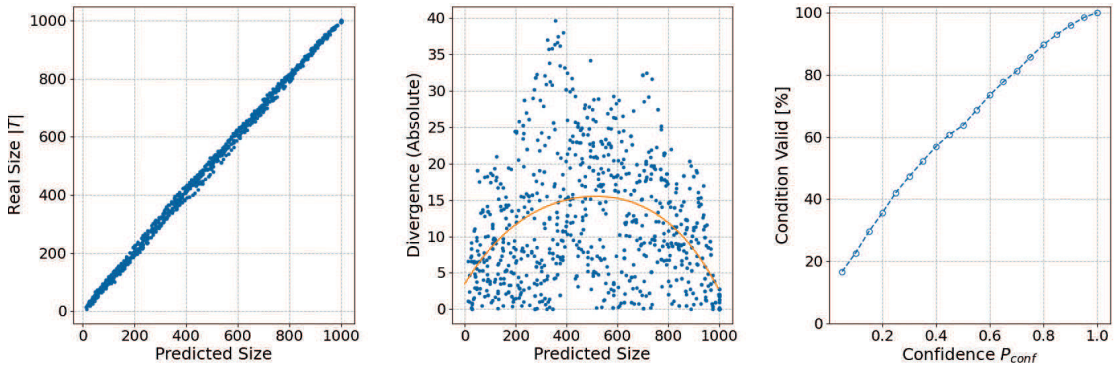
4.6.4 Flexible Product Quantization

Flexible product quantization (Section 4.5.6) can adjust the product quantization distance calculation to smaller or larger sets of vectors. The product quantization sequences in our index structure for the Google News dataset consist of codes $c_i \in \{0, \ldots, 31\}$ of length 30, which can be combined into shorter codes $c'_i \in \{0, \ldots, 1023\}$ with length 15. The overall execution time of a kNN-Join with product quantization distance calculation is



(a) Execution Times of Both Methods (b) Precomputation and Distance Computation

Figure 4.19: Evaluation of Short and Long Codes Calculation



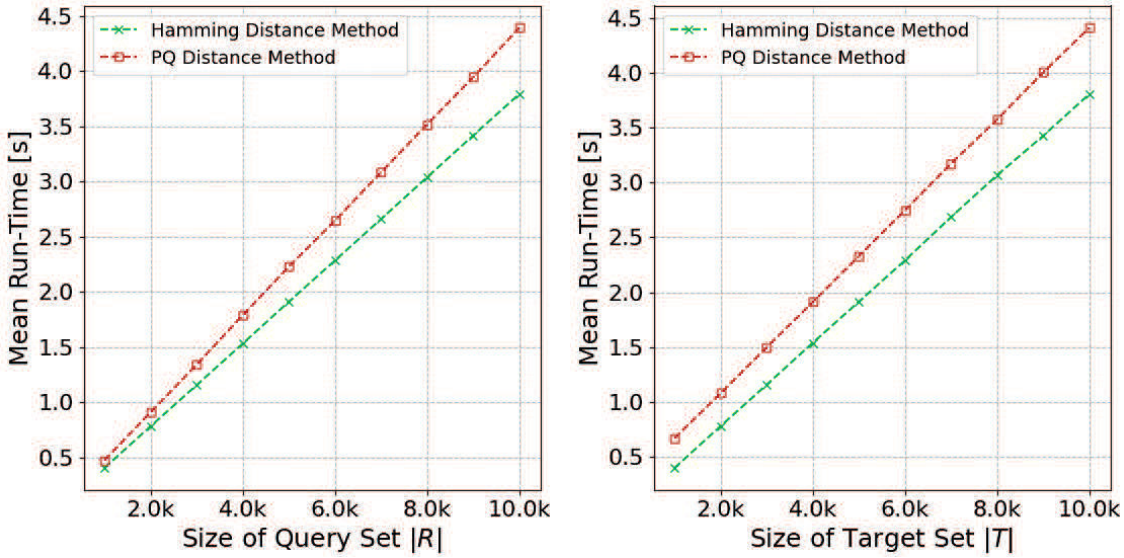
(a) Estimated Size Values (b) Divergence (Absolute) (c) Confidence

Figure 4.20: Estimation of Target Set Size

shown for both methods in Figure 4.19a. Figure 4.19b shows the execution times for the precomputation and the distance calculation step. We use query sets of size $5,000$. The target set size $|T|$ is shown on the x-axis. The value α is set to $\frac{|T|}{2 \cdot k}$. The measurements are done 10 times with randomly sampled query and target vectors, and average values are determined. For small target sizes with $|T| \leq 20,000$, the computation via long codes is faster. For larger target sets, the overhead of the distance calculation for long codes becomes prevalent such that the calculation with short codes is faster.

4.6.5 Accuracy of the Target Size Estimation

The number of targets determined in the retrieval step of the algorithm before the distance calculation can be estimated. For this purpose, we leverage an approximation of the hypergeometric distribution as described in Section 4.5.5. The estimated number of targets derived from the index is μ, as defined in Equation (4.3). In Figure 4.20a, a scatter



(a) Increasing Query Size ($|T|= 10,000$) (b) Increasing Target Size ($|R|= 10,000$)

Figure 4.21: Performance of Word2Bits kNN-Join Algorithms

plot of the estimated and actually derived number of targets is shown. For these measurements, kNN-Joins with a single randomly sampled query vector are executed, and the number of targets obtained in the first retrieval step is determined inside the user-defined function. This is done for all $\alpha \in \{1, \ldots, 100\}$, $k = 10$, and target sets of size $|T|= 1,000$. For each α value, 10 queries are executed. The divergence of the estimation is higher if the desired number of targets per query vector gets higher. This can be noticed in the 4th-grade polynomial regression curve of the sample points in Figure 4.20b. However, if the number of desired targets is near $|T|$, it is apparently decreasing.

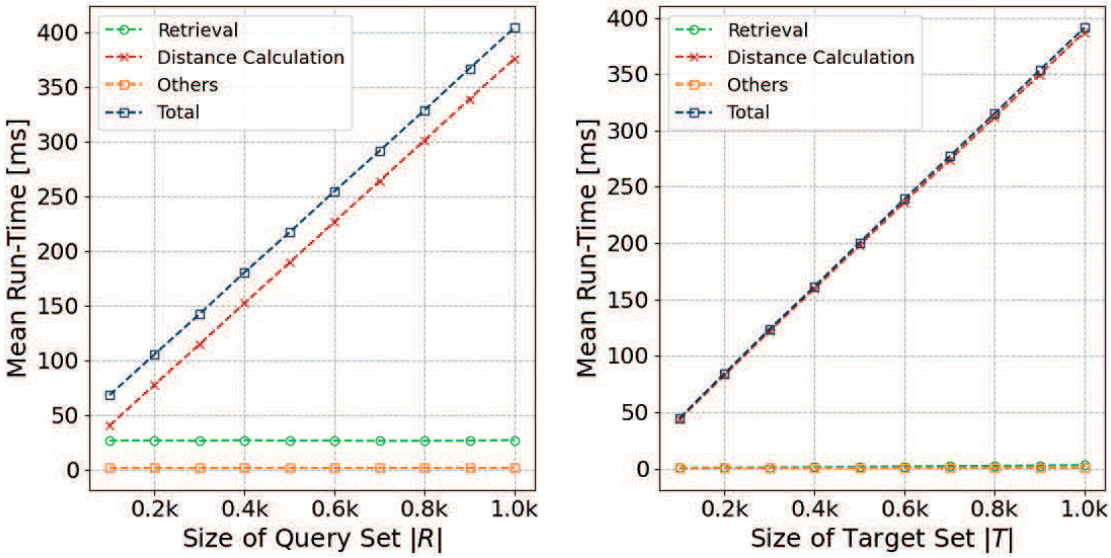
To prevent the system from executing lots of database queries, one can adjust the confidence value P_{conf}. It represents how likely it is that only one database request is sufficient to derive the desired number of targets from the index. This is also evaluated by single query kNN-Joins with $\alpha = 10$ and the same search parameters as in the last experiment. The number of queries where the condition is satisfied (only one request is required) in relation to P_{conf} is shown in Figure 4.20c. For each confidence value $P_{conf} \in \{0.05 \cdot i | i = 1, \ldots, 20\}$, 1,000 queries are executed. As desired, the amount of queries where the condition is fulfilled rises to 100% if the confidence value increases up to 1. However, the actual confidence is higher than P_{conf} since the confidence can only be increased step-wise by incrementing ω.

4.6.6 Performance of Word2Bits kNN-Join

We evaluate the performance of our implementation of kNN-Joins for word2bit vectors via the Hamming distance calculation. Therefore, we compare the performance of this calculation with a kNN-Join implementation using the exhaustive product quantization search. Further, we investigated the run-time of the subroutines of this implementation.

Dataset: For our evaluation, we used a dataset[27] of $400,000$ 800-dimensional word2bit vectors trained on Wikipedia. We already presented the data characteristics of this embedding model in Section 3.2.

[27] https://drive.google.com/open?id=107guTTy93J-y7UCO2ZA2spxRIFpoqhjh (Access: 03/31/21)



(a) Increasing Query Size ($|T|= 10,000$)

(b) Increasing Target Size ($|R|= 10,000$)

Figure 4.22: Performance of Word2Bits kNN-Join Subroutines

Performance of Bit-wise Calculation: We evaluate the performance of our word2bits specific kNN-Join algorithm with respect to increasing sizes of the target set $|T|$ and the query set $|R|$. We choose the base-wise product quantization kNN-Join implementation as a baseline. This algorithm also serves as a baseline in the experiments described in Section 4.6.2. The product quantization index created for this purpose consists of $m = 40$ quantization functions with $|C|= 32$ centroids. For each parameter setting, we execute the kNN-Join algorithms 10 times and determine the mean run-time.

The results of this evaluation are shown in Figure 4.21. As one can see, the exact kNN-Join algorithm based on the Hamming distance calculation outperforms the approximated algorithm based on the product quantization index in all situations investigated here. Moreover, Figure 4.21a shows the absolute performance difference increases with increasing query size. A similar effect is observed in Figure 4.21b for increasing target sizes. However, in this scenario, the difference in the run-time measurements increases only slightly. Accordingly, the relative difference between the run-time measurements is much higher for small target sizes.

Performance of Subroutines: The word2bits kNN-Join algorithm retrieves all target vectors as binary representations from the vector table in the database. Afterward, the distance calculation is performed to obtain the k nearest neighbors. We evaluate the run-time of the two subroutines while increasing the size of the query and the target set. Therefore, we execute 10 kNN-Join queries for each data point depicted in Figure 4.22. Figure 4.22a shows the mean run-time values for query sizes $|R|$ between 100 and $1,000$, where the target set contains $10,000$ vectors. In contrast, Figure 4.22b visualizes the mean run-time values for target sizes $|T|$ between 100 and $1,000$, where the query set encompasses $10,000$ vectors. As one can see, for large query sets R, the distance calculation is the most comprehensive subroutine. However, if the query set is smaller, the vector retrieval step also requires a significant amount of the total execution time. Other subroutines do not demand a significant amount of run-time in all the experiments demonstrated here.

4.7 SUMMARY

In this chapter, we presented our integration of word embedding operations into relational database systems. This lead to the development of the PostgreSQL extension FREDDY (Section 4.1). Thereby, various novel query types are enabled (see Section 3.4). We implemented them as UDFs (see Section 4.1.3) to allows the user to combine them with the functionality of SQL. This enables the user, for instance, to apply standard SQL operations on records obtained by word embedding operations based on the semantic similarity or relatedness of their attributes to certain text values. Moreover, the novel operations are independent of a specific embedding model. Thus, they can utilize different word embedding datasets which have been imported into the database. This enables the user to switch between different notions of similarity. In addition, a Web application (see Section 4.1.4) serves as a user-friendly interface for executing word embedding SQL queries.

Since most of the word embedding operations can be executed via kNN-Joins (see Section 4.1.3), we proposed in Section 4.5 an ANN-Join algorithm to increase the efficiency of the operations. To achieve this, we surveyed related work on nearest neighbor search in Section 4.2. Based on our analysis of the applicability of those techniques for ANN-Joins of word vectors (see Section 4.3), we decided to build the design of our algorithm upon state-of-the-art vector quantization techniques for approximated nearest neighbor search. Our algorithm implements two novel optimizations: candidate number estimation (see Section 4.5.5) and flexible product quantization (see Section 4.5.6). In contrast to related work (see Section 4.4), our algorithm can efficiently execute kNN-Join tasks with a large query set, as well as queries with a large target set. Therefore it is well-suited for the different kinds of kNN-Join tasks performed during the execution of word embedding operations. The results of our evaluation in Section 4.6 show the effectiveness of those optimizations and the efficiency of the search algorithm in general. In addition, we propose an algorithm specifically for kNN-Joins on word2bits embeddings in Section 4.5.9 which also shows its effectiveness in the evaluation in Section 4.6.6.

5

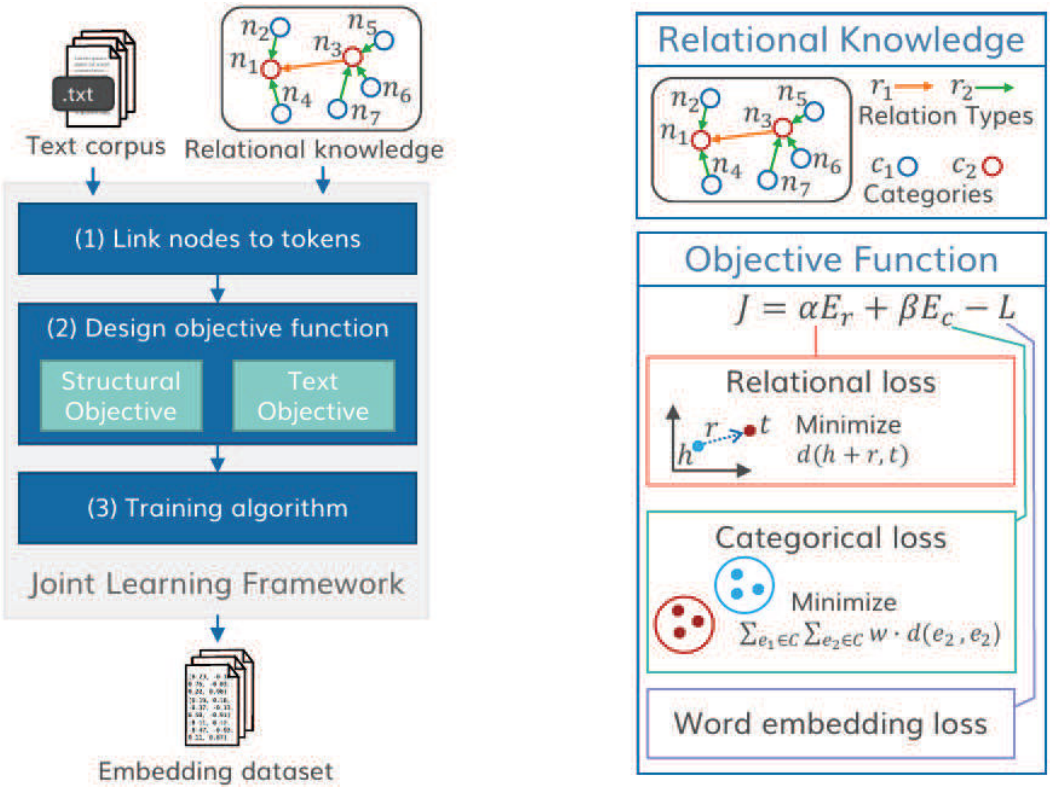
CONTEXT ADAPTATION FOR WORD EMBEDDING OPTIMIZATION

In this chapter, we survey techniques for optimizing embedding representations of text values by utilizing additional structured data. Based on this, we introduce a novel context adaptation algorithm called RETRO to optimize embeddings of text values in database systems by exploiting relations between text values in relational database systems, as stated in Objective C in Section 3.3. To cope with the demands of a context adaptation process in relational database systems, we design this algorithm to satisfy the requirements stated in Section 3.6. Our research on the relational retrofitting framework RETRO has led to several publications [GTL19b, GTL20, GTNL20, GOTL20] on which this chapter is based.

In the first Section 5.1, we discuss related work. In Section 5.2, we describe our relational retrofitting approach RETRO. Afterward, we introduce the evaluation platform RETRO-LIVE in Section 5.3. It follows a comprehensive evaluation of the proposed techniques in Section 5.4. Finally, we summarize our contribution and findings presented in this chapter in Section 5.5.

5.1 RELATED WORK

Several methods have been proposed to combine the abilities of word embedding models with the relational knowledge of a structured data source. Those methods can be broadly categorized in *joint embedding methods* described in the following Section 5.1.1 and *retrofitting approaches* introduced in Section 5.1.2. Moreover, we discuss the applicability of table embedding models for this purpose if the structured data source constitutes a set of tables in Section 5.1.3. While joined embedding approaches design embedding models to be trained directly on texts and relational data in a joint manner, retrofitting approaches constitute post-processing methods, which adjust an already pre-trained embedding model by exploiting relational data.



(a) Joint Embedding Approach

(b) RC-Net Algorithm

Figure 5.1: Joint Embedding Methods

5.1.1 Graph and Text Joint Embedding Methods

Joint embedding models consider relational information in the form of property graphs and word co-occurrences obtained from text documents for training word embedding models. Therefore, those models design an objective function that takes both aspects into account. A wide range of different joint embedding models has been proposed [WZFC14, XBB+14, YD14, ZZW+15, BAMK16]. Figure 5.1a shows the typical architecture of those joint embedding algorithms. First, the models need to (1) identify links of tokens in the text documents to nodes in the graph representing the relational knowledge. Those links are used later in the embedding training to model the connection between the tokens and the nodes. A simple way of linking is to check for syntactic equivalence. However, this could lead to wrong and missing links since words could be ambiguous. Specifically, named entities frequently have multiple names. Therefore, several alternative linking approaches have been developed. The authors of [WZFC14] propose to use links of named entities provided by the data sources[1], or employing an entity linking system to identify links between the knowledge graph and the text documents. Alternatively, [ZZW+15] proposes to link the words in additional entity descriptions to nodes in the knowledge graph representing the entities. After the linking process, the training objective is defined (2), which usually consists of a textual loss referring to properties of the text and a structural loss referring to the connections in the knowledge graph. Multiple models [XBB+14, WZFC14, YD14, ZZW+15] use loss functions similar to the functions used in the word2vec Skip-Gram or CBOW model as textual loss function and add a structural

[1]The authors used a text corpus of Wikipedia articles and exploit the fact that the labels of hyperlinks often refer to entities in Freebase which correspond to the nodes in the knowledge graph.

loss to treat embeddings nearby if they refer to objects related in the knowledge graph. While some approaches [XBB+14, YD14] only manage one set of embeddings with links assigned in step (1), other embedding models [WZFC14, ZZW+15] initialize two separate sets of embeddings for tokens and nodes in the graph. For the links between both sets of embeddings, an additional loss function models the alignment in those approaches. Various structural loss functions have been proposed to model different relation types. In [YD14, BAMK16], the authors define structural loss functions, which consider entities either as related or not related. The approaches in [WZFC14, ZZW+15] support different relation types. Therefore, a knowledge graph is modeled as a set of triples, and a structural loss is used, which is similar to the loss of the graph embedding technique TransE [BUGD+13]. Moreover, in [XBB+14] relational knowledge in the form of categorical and relational connections is supported. After the loss function is designed, the training algorithm (3) is applied. Usually, a variant of the stochastic gradient descent algorithm is used to train a neural network similar to the training of learned embedding models (see Section 2.2.2). To generate training data, different methods are possible. Some joint learning approaches like RC-Net [XBB+14] generate training data by processing the text documents. Thereby, the structural loss is calculated for nodes liked to tokens which are traversed. In this way, the structural loss acts as a regularization function for the neural network model. Alternatively, approaches like pTransE [WZFC14] simultaneously generate training samples from the knowledge graph to optimize the structural loss and word sequences from the text to optimize the textual loss.

Recently, several approaches [PNL+19, ZHL+19] have been proposed to integrate relational knowledge into the training process of contextualized word embedding models. Those models are designed for training on a text corpus with entity mentions linked to entities in a knowledge base. Therefore, the proposed methods train knowledge graph embeddings for the entities, e.g., in [ZHL+19], the authors use TransE for this purpose. Those static embeddings are used as additional inputs for a BERT model, which is extended by layers to integrate them with the embeddings of aligned tokens. For training the model, in [ZHL+19], a new pre-training objective was designed. Here, the alignment of entities to their corresponding tokens is masked and should be predicted by the model.

In the following, we describe the joint learning method RC-Net [XBB+14] in detail since it is, in contrast to the other models, expressive enough to model the different types of relations occurring in a relational database system (see Requirement C2 in Section 3.6).

RC-Net The model proposed by [XBB+14] considers data in the form of a text corpus and a knowledge graph with relational and categorical connections. For RC-Net, no specific method is proposed to link entities from the knowledge base to tokens in the text corpus. In the evaluation in [XBB+14], the authors derive the linking from syntactic equivalences between the labels in the knowledge base and the words in the training corpus. The objective function constitutes a joint loss with three parts: a Skip-Gram loss, a relational loss, and a categorical loss. Figure 5.1b visualizes the objective function.

Relational loss: The relational loss is inspired by TransE [BUGD+13], which models relations by translation operations of embedding representations. Thereby, a relation is defined by a triple $(h, r, t) \in S$ of two tokens $h, t \in V$ and a relation $r \in R$ which is obtained from the knowledge resource. The tokens h and t correspond to the word embeddings $\mathbf{v_h}$ and $\mathbf{v_t}$, and the relation r is assigned to a relation embedding $\mathbf{v_r}$. If the relation (h, r, t) holds, the distance $d(\mathbf{v_h} + \mathbf{v_r}, \mathbf{v_t})$ should be small. During the training of the model, $d(\mathbf{v_h} + \mathbf{v_r}, \mathbf{v_t})$ should be minimized if (h, r, t) is a valid relation and maximized if (h, r, t) is invalid. Therefore, the authors define the relational loss function E_r:

$$E_r = \sum_{\substack{(h,r,t) \\ \in S}} \sum_{\substack{(h',r,t') \\ \in S'_{(h,r,t)}}} [\gamma + d(\mathbf{v_h} + \mathbf{v_r}, \mathbf{v_t}) - d(\mathbf{v_{h'}} + \mathbf{v_r}, \mathbf{v_{t'}})]_+ \tag{5.1}$$

Here, $S'_{(h,r,t)}$ denotes a set of corrupted triples:

$$S'_{(h,r,t)} = \{\{(h', r, t)|h' \in V\} \cup \{(h, r, t')|t' \in V\}\} \setminus S \tag{5.2}$$

In Equation (5.1), $\gamma > 0$ is a hyperparameter, $d(x, y)$ denotes the Euclidean distance, and $[x]_+ = max(x, 0)$ the positive part of x. A naïve way to optimize E_r is to maximize the norm of the embedding representations. To prevent this, a soft-norm constraint is applied on the relational embeddings that forces each dimension r_i of an embedding $\mathbf{v_r}$ to be within the range $(-1, 1)$. To efficiently calculate the loss, only a sample of triples is drawn from S'

Categorical loss: The categorical loss function treats the embeddings of text values which are part of the same group to be similar. To do this, the authors propose to define for each pair of tokens (w_i, w_j) a similarity score $s(w_i, w_j)$ that is non-zero if both tokens belong to at least one common category. The score $s(w_i, w_j)$ is higher if the two tokens belong to small categories with only a few instances and lower if the terms only belong to a broad category. The scores should be defined in a way that all similarity scores of a token w_i sum up 1:

$$\forall w_i \in V : \sum_{w_j \in V} s(w_i, w_j) = 1 \tag{5.3}$$

The categorical loss E_c itself is defined by the following equation:

$$\sum_{w_i \in V} \sum_{w_j \in V} s(w_i, w_j) d(\mathbf{v_i}, \mathbf{v_j}) \tag{5.4}$$

Joint Loss: The joint loss function combines E_r, E_c, and the Skip-Gram loss function L, which should be maximized into a single loss function:

$$J = \alpha E_r + \beta E_c - L \tag{5.5}$$

Thereby, the hyperparameters α and β constitute weights for the relational and the categorical loss.

Training Algorithm: The training of the embedding model resembles the training of the Skip-Gram model. Thus, the training is done by processing word context pairs by a neural network (see Section 2.2.2). The additional loss functions E_r and E_c serve as regularization functions. For optimizing the ANN model, a stochastic gradient descent algorithm is used.

Applicability for Context Adaptation In Section 3.6, we defined several requirements for a method for a context adaptation approach to optimize the embedding representations of text values in database systems. Requirement C1 stated that the adaptation should be holistic in the sense that it can capture the knowledge from structural and textual knowledge resources. Joint learning models are effective approaches to construct embedding models with these capabilities.

Several different kinds of relations between text values can be extracted from a relational database system. Requirement C2 expects from an adaptation algorithm enough expressiveness to model those relations. Most of the relations of text values in DBMSs can effectively be modeled by categorical and relational connections. So principally, RC-Net provides the desired expressiveness. However, RC-Net only allows generating one embedding for multiple syntactical equivalent text values. In a database, the same text value

might occur in several tables and carry different semantic properties. Moreover, a text value in a database system is not equivalent to a token in a text. RC-Net can not handle such cases.

Another important feature for the desired optimization algorithm is updatability (Requirement C3). Joint models require to process text during training. After the training process, they can not react to changes in the structural knowledge resource and update the embedding representations accordingly. Thus online updates (Requirement C3) are not possible. According to Requirement C4, the context adaptation should be integrated into a database system. This is not the case for any of the joint learning algorithms.

5.1.2 Retrofitting Approaches

Retrofitting approaches expect an already existing set of embedding representations which they adjust by incorporating additional structured knowledge in a post-processing manner.

Original Retrofitting The original retrofitting model [FDJ+15] proposed by Faruqui et al. takes word embeddings in a matrix W^0 and a graph $G = (Q, E_F)$ representing a lexicon as input [FDJ+15]. The retrofitting problem is formulated as a dual objective optimization function: The embeddings of the matrix W are adapted by placing similar terms connected in the graph G closely together while at the same time the neighborhood of the terms from the original matrix W^0 should be preserved. Here, $Q = \{q_1, \ldots, q_n\}$ is a set of nodes where each node q_i corresponds to a word vector $\boldsymbol{v_i} \in W^0$ and $E_F \subset \{(i,j)|i,j \in \{1,\ldots,n\}\}$ is a set of edges. The graph is undirected, thus $(i,j) \in E_F \Leftrightarrow (j,i) \in E_F$. The authors specified the retrofitting problem as a minimization problem of the following loss function:

$$\Psi_{Faruqui}(W) = \sum_{i=1}^{n} \left[\alpha_i ||\boldsymbol{v_i} - \boldsymbol{v_i'}||^2 + \sum_{j:(i,j)\in E_F} \beta_i ||\boldsymbol{v_i} - \boldsymbol{v_j}||^2 \right] \tag{5.6}$$

The constants α_i and β_i are hyperparameters. $\Psi_{Faruqui}(W)$ is convex for positive values of α_i and β_i. Thus, the optimization problem can be solved by an algorithm, which iterates over every node in Q and updates the respective vector in W according to Equation (5.7). The update function for any vector $\boldsymbol{v_i}$ is obtained from the root of the partial derivative $\frac{\partial \Psi_{Faruqui}(W)}{\partial \boldsymbol{v_i}}$.

$$\boldsymbol{v_i} = \frac{\alpha_i \boldsymbol{v_i'} + \sum\limits_{j:(i,j)\in E_F} (\beta_i + \beta_j)\boldsymbol{v_j}}{\alpha_i + \sum\limits_{j:(i,j)\in E_F} (\beta_i + \beta_j)} \tag{5.7}$$

However, Faruqui et al. decided to use this simplified update function:

$$\boldsymbol{v_i} = \frac{\alpha_i \boldsymbol{v_i'} + \sum\limits_{j:(i,j)\in E_F} \beta_i \boldsymbol{v_j}}{\alpha_i + \sum\limits_{j:(i,j)\in E_F} \beta_i} \tag{5.8}$$

While there is no justification in the paper, the update function in Equation (5.8) gets along with some advantages. The update function in Equation (5.7) can shift embeddings strongly to prevent an adaptation of its neighboring nodes. This leads to a higher

loss in the summand of this single vector but lowers the value of the global loss function. This might not be advantageous since the result of a classification task might depend on a few embeddings and not on the whole embedding corpus. Moreover, Equation (5.8) is slightly faster to compute. In contrast to Equation (5.7), using the update function of Equation (5.8), the algorithm does not converge to a global minimum. Further, using this equation, the results of the retrofitting depend on the order in which the vectors are updated as already observed by [SC16]. The calculation using updates as in Equation (5.8) does not solve the optimization problem in Equation (5.6), however, can be considered as a formulation of a recursively defined series. For every $\boldsymbol{v_i} \in W$, the series converges to a specific vector which is the desired output of the retrofitting process.

Retrofitting Adaptations Several papers proposing different adaptations of the model proposed by Faruqui et al. [FDJ+15] for specific tasks and data sources: In [KHC15], the authors propose a model to specialize a word embedding model to either capture similarity or relatedness. [MST+16] investigates retrofitting methods that disambiguate between similar and dissimilar relations. Functional relations of property graphs can be retrofitted as proposed by [LMP18]. While Faruqui et al. update the vectors one by one, we used a matrix formulation that updates whole vectors at once. A similar matrix update for retrofitting was first proposed by [SC16].

Applicability for Context Adaptation Like the joint learning methods retrofitting approaches also capture the knowledge from structural and textual knowledge resources and thus fulfill Requirement C1 of Section 3.6. In general, an adaptation via retrofitting is faster and requires lower resource consumption than the training of a new embedding model as done by the joint embedding approaches. According to the experiments in [FDJ+15], retrofitting can improve word embeddings according to intrinsic tasks better than the joined learning approach proposed by [YD14], which also favors a post-processing approach.

As far as we know, there is no retrofitting approach that is expressive enough to capture all the different relation types which exist between text values in database systems. Thus, Requirement C2 is not satisfied.

Requirement C3 demands an online updatable version of the retrofitting algorithm. Simple retrofitting approaches like the original approach are very fast and allow retrofitting of hundreds of thousands of vectors in only a few seconds on ordinary consumer hardware (see Section 5.4.5). However, the run-time of the algorithm increases linearly with the size of the dataset, and in the case of frequent updates, a few seconds are still unacceptable. Unfortunately, existing approaches do not allow online updates.

The integration of retrofitting algorithms into relational database systems has not been a subject of research. Therefore, Requirement C4 is not satisfied.

5.1.3 Table Embedding Models

Besides using models to incorporate relational information in the form of a graph into embedding representations, it is also possible to directly train embedding models on tabular data. Multiple static [GRE+17, ZZB19] and contextualized word embedding models [DSL+20, YNYR20] have been proposed for this purpose. In Chapter 7, we give a detailed overview of those methods and their applications. Several of these techniques like TABERT [YNYR20] allow joint training on a text corpus and a corpus of tables. In

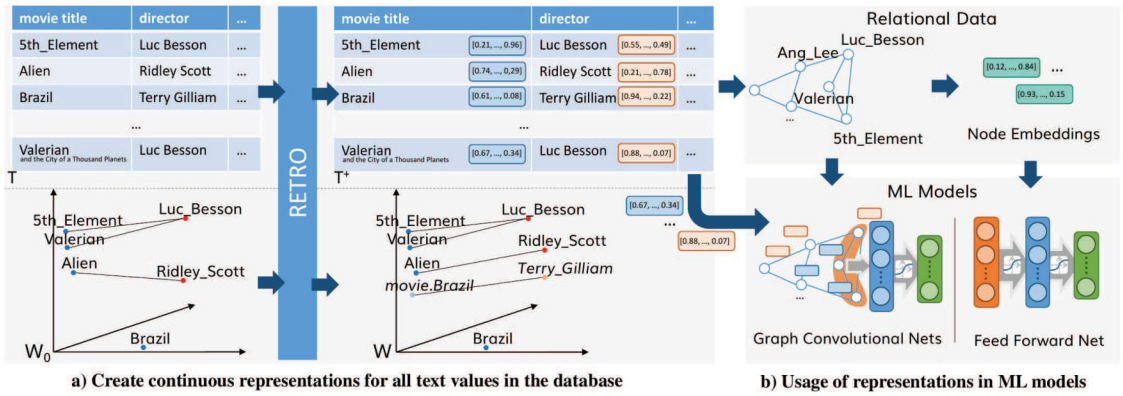


Figure 5.2: Relational Retrofitting: base embeddings W^0 and relation T, retrofitted embedding W and augmented relation T^+

this way, those techniques combine knowledge from (semi-) structured and unstructured textual information. However, those techniques aim at training an embedding model by using large corpora of tables, e.g., in the evaluation in [YNYR20], the authors used 26 million tables for training TABERT. In contrast, our goal is an adaptation of embedding representations of text values to the local domain-specific database, which usually contains a comparable small set of tables. Thus, it is not clear how those techniques perform on such comparable small sets of tables. Moreover, most of the limitations mentioned for the joint learning methods in Section 5.1.1 also apply to table embedding models. Nevertheless, one can execute post-processing techniques like RETRO on embedding representations generated by table embedding models for the text values in the database. In this way it is potentially possible to further improve embedding representations generated by table embedding models.

5.2 RELATIONAL RETROFITTING APPROACH

Based on our review of context adaptation methods above in Section 5.1, we decided to use a retrofitting method to optimize embedding representations of text values in database systems. However, since previously developed retrofitting approaches do not satisfy all the requirements for an application on relational database systems stated in Section 3.6, we design a novel retrofitting framework called RETRO[2] [GTL20] for this purpose. We build upon the idea of the original retrofitting approach proposed by [FDJ+15] and our proposed optimization model can be viewed as a generalization of this approach.

Figure 5.2a depicts the application of RETRO on a database system. Therefore, it utilizes the information given by the disposition of the text values in the database schema, e.g., which values appear in the same column or are related, to improve the embedding representation. RETRO is able to automatically derive high-quality numerical representations of textual data residing in databases without any manual effort. Specifically, the framework pursues the following three opportunities for optimizing embedding representations of text values in the database:
1) It combines word embedding knowledge with relational information by moving vectors in the direction of text value vectors that are related or appear in the same column. In the example, the embedding for "Valerian" will be close to other movie titles and close to its director "Luc Besson". In this way, it is more similar to movies of French directors,

[2] https://github.com/guenthermi/postgres-retrofit (Access: 08/26/21)

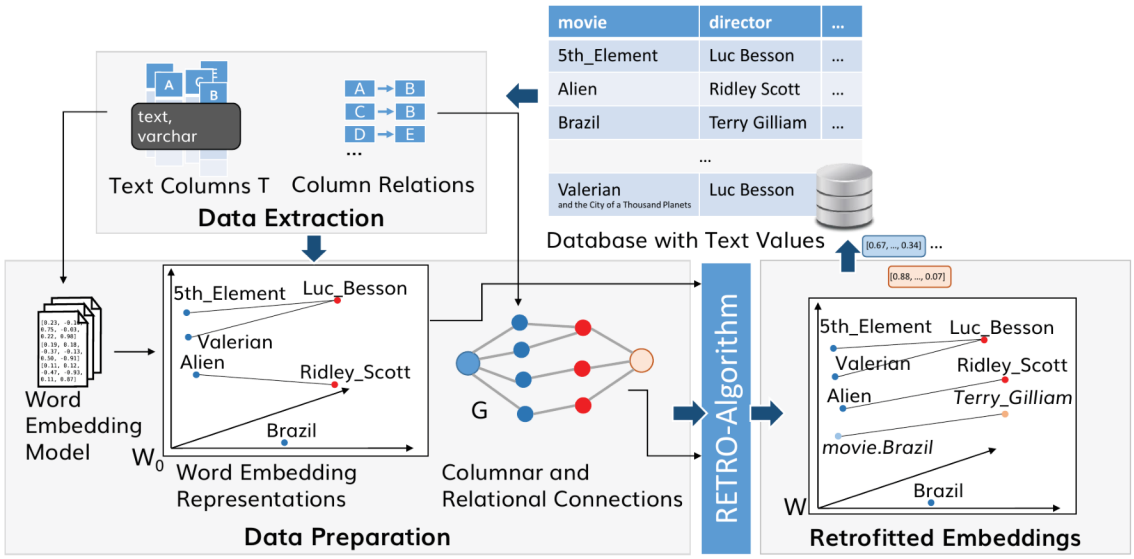


Figure 5.3: Overview of the RETRO Framework

which might be beneficial for information retrieval and ML tasks.
2) Performing those adaptations, RETRO generates unique representations for every distinct text value in the database. Thereby, ML models are qualified to distinguish the national soccer team "Brazil" from the sci-fi movie with the same name.
3) It integrates the semantic of related text values into the text value embeddings. In this way, ML models can recover information about properties of related text values not prevalent in the embedding itself. For instance, from a movie description column in the database, it could be derived that "Valerian" is based on a comic which is then also represented in the movie title embedding. This information would not be available if word embedding and relational information are considered separately.

Overview of the RETRO Framework Figure 5.3 gives an overview of the RETRO framework. It starts with analyzing the schema of the relational database system. Specifically, it identifies columns with text values and relations between those columns. Two columns are considered related if they are either part of the same table or foreign key relations connect the tables of the columns. Then, for each text column, its values and their relations to text values in related columns are extracted. This process is described in Section 5.2.1. Afterward, an optimization problem described in Section 5.2.2 is solved by the relational retrofitting algorithm (see Section 5.2.3), and the resulting vectors are stored in the database. To generate embedding representations of text values added later to the database, we propose an algorithm for online updates in Section 5.2.4.

Applications Retrofitted embeddings of text values in database systems can be utilized in various applications. Supervised machine learning models constitute one field of applications. Since it is relatively easy to evaluate such ML models with respect to the embedding model used to encode the input, we use them to determine the effectiveness of our retrofitting approach. To allow the ML models to take the relations of text values in the database into account, we provide them either directly with a graph of text value relations or in the form of additional node embeddings trained on the graph as shown in Figure 5.2b. On the one hand, we implemented simple feed-forward neural networks for various tasks. On the other hand, we developed a graph-neural network model for missing value imputation, which operates on the embedding representations of text values in the database. Section 5.4.3 provides more details about the implementation of the models.

5.2.1 Data Preparation

RETRO learns a matrix of vector representations $W = (\boldsymbol{v_1}, \ldots \boldsymbol{v_n})$ with $\boldsymbol{v_i} \in \mathbb{R}^D$ for every text value $T = (\boldsymbol{t_1}, \ldots \boldsymbol{t_n})$ in a cell of a database. Multiple syntactically equivalent text values in a column obtain one common embedding, however, syntactically equivalent text values occurring in different columns are assigned to separate embedding representations. The matrix W is initialized with embedding representations obtained by using a pre-trained embedding model. In addition, relations are extracted and compiled into a graph G.

Embedding Vector Initialization

The optimal method for the initialization of the embedding vectors depends on the embedding models. We apply RETRO to embeddings generated with static word embedding models like Word2Vec and GloVe and embeddings of a BERT-based model, which is fine-tuned to produce static embedding representations.

Static Word Embedding Models: To find initial embedding vectors for every text value, we tokenize the text values based on the vocabulary of the word embedding model and average those vectors to obtain representations for complete database text values. This procedure [ALM17], described in Section 2.2.5, is convenient to represent short phrases [AMCG15, ZWX15]. Since large word embedding datasets like the Google News word2vec dataset (W2V-GN in Table 3.1) contain mainly multi-words and phrases, it is not trivial to find the optimal tokenization. To take into account multi-word tokens, we propose the following tokenization approach: First, a lookup trie (prefix tree) is created for the dictionary of the given word embedding dataset, where every node represents a token. By considering the lookup trie the longest possible sequences of nodes are extracted, e.g., "bank account" instead of "bank", resulting in a bag of tokens for each text value. Text values consisting only of unknown tokens initially get assigned a null vector. However, since RETRO is also considering the relations to other text values, these values get assigned a meaningful representation in the learning phase. Finally, a matrix $W^0 = (\boldsymbol{v'_1}, \ldots \boldsymbol{v'_n})$ stores all embedding representations, forming the basis for the retrofitting process.

Contextualized Word Embedding Models: One can also use a BERT-based model such as SentenceBert [RG19] to directly obtain a set of embeddings for text values. SentenceBert produces static embeddings for phrases to solve NLP tasks very efficiently by calculating cosine similarity values. We use this method to get initial embeddings for W^0.

Extracting Data from the Database

To capture the semantic relations between text values, we extract *columnar and relational connections* from the relational schema. Those connections are used by RETRO to create a representation capturing the context of the text value in the database and thus help to preserve their semantic more accurately compared to a plain word embedding representation.

Columnar Connections: Text values appearing in the same column usually form hyponyms of a common hypernym (similar to subclass-superclass relations). Thus, they share a lot of common properties, which typically lead to similarity. We capture this information and assign each text value t_i to its column $C(i)$.

Relational Connections: Relations exhibit from the co-occurrence of text values in the same row and from foreign key relations. Those relations are important to characterize the semantic of text value in the database. We define a set of relation types R for each specific pair of related text value columns. Those columns are related because they are either part of the same table or there exists a foreign key relationship between their tables. For every relation type $r \in R$, there is a set E_r containing the tuples of related text value ids. Relation types are directed.

Graph Generation: Columnar and relational connections can also be compiled in a graph $G = (V, E)$. The node-set $V = V_C \cup V_T$ consists of nodes V_T for every distinct text value in a database column and blank nodes for every column V_C. The edge set $E = \bigcup_{r \in R} E_r \cup E_C$ consists of a set of edges E_r for every relational type and edges E_C connecting text values of one column to a common column node.

5.2.2 Relational Retrofitting Problem

RETRO considers relational and columnar connections (see Section 5.2.1) to retrofit an initial embedding. Accordingly, we define a *loss function* Ψ adapting embeddings to be similar to their original word embedding representation W^0, the embeddings appearing in the same column (columnar loss Ψ_C), and related embeddings (relational loss Ψ_R).

$$\Psi(W) = \sum_{i=1}^{n} \Big[\alpha_i ||\boldsymbol{v_i} - \boldsymbol{v_i'}||^2 + \beta_i \Psi_C(\boldsymbol{v_i}) + \Psi_R(\boldsymbol{v_i}, W)\Big] \tag{5.9}$$

The *columnar loss* is defined by Ψ_C and treats every embedding $\boldsymbol{v_i}$ to be similar to the constant centroid $\boldsymbol{c_i}$ of the original embeddings of text values in the same column $C(i)$.

$$\Psi_C(\boldsymbol{v_i}) = ||\boldsymbol{v_i} - \boldsymbol{c_i}||^2 \quad \boldsymbol{c_i} = \frac{\sum\limits_{j \in C(i)} \boldsymbol{v_j'}}{|C(i)|} \tag{5.10}$$

The *relational loss* Ψ_R treats embeddings v_i and v_j to be similar if there exists a relation between them and dissimilar otherwise. E_r is the set of tuples where a relation r exists. $\widetilde{E_r}$ is the set of all tuples $(i, j) \notin E_r$ where i and j are part of relation r. Thus, each of both indices has to occur at least in one tuple of E_r.

$$\Psi_R(\boldsymbol{v_i}, W) = \sum_{r \in R} \Big[\sum_{\substack{j:(i,j) \\ \in E_r}} \gamma_i^r ||\boldsymbol{v_i} - \boldsymbol{v_j}||^2 - \sum_{\substack{k:(i,k) \\ \in \widetilde{E_r}}} \delta_i^r ||\boldsymbol{v_i} - \boldsymbol{v_k}||^2 \Big] \tag{5.11}$$

α, β, γ, and δ are hyperparameters. Ψ should be a convex function. We proved in Appendix A that convexity is assured if the hyperparameters fulfill the following inequalities:

$$\begin{aligned} &\forall r \in R, i \in \{1, \ldots, n\} \;\; (\alpha_i \geq 0, \;\; \beta_i \geq 0, \;\; \gamma_i^r \geq 0) \\ &\forall \boldsymbol{v_i} \in W \;\; (4\alpha_i - \sum_{r \in R} \sum_{j:(i,j) \in \widetilde{E_r}} \delta_i^r \geq 0) \end{aligned} \tag{5.12}$$

In practice, however, other parameter configurations that do not comply might work as well. Given the property of convexity, an iterative algorithm can be used to minimize Ψ. This algorithm iteratively executes for all $\boldsymbol{v_i} \in V$ the following equation, which is derived from the root of the partial derivative $\frac{\partial \Psi(W)}{\partial \boldsymbol{v_i}}$.

$$\boldsymbol{v_i} = \frac{\alpha_i \boldsymbol{v_i'} + \beta_i \boldsymbol{c_i} + \sum_{r \in R} \Big[\sum\limits_{\substack{j:(i,j) \\ \in E_r}} (\gamma_i^r + \gamma_j^{\bar{r}}) \boldsymbol{v_j} - \sum\limits_{\substack{k:(i,k) \\ \in \widetilde{E_r}}} (\delta_i^r + \delta_k^{\bar{r}}) \boldsymbol{v_k} \Big]}{\alpha_i + \beta_i + \sum_{r \in R} \Big[\sum\limits_{\substack{j:(i,j) \\ \in E_r}} (\gamma_i^r + \gamma_j^{\bar{r}}) - \sum\limits_{\substack{k:(i,k) \\ \in \widetilde{E_r}}} (\delta_i^r + \delta_k^{\bar{r}}) \Big]} \tag{5.13}$$

5.2.3 Relational Retrofitting Algorithm

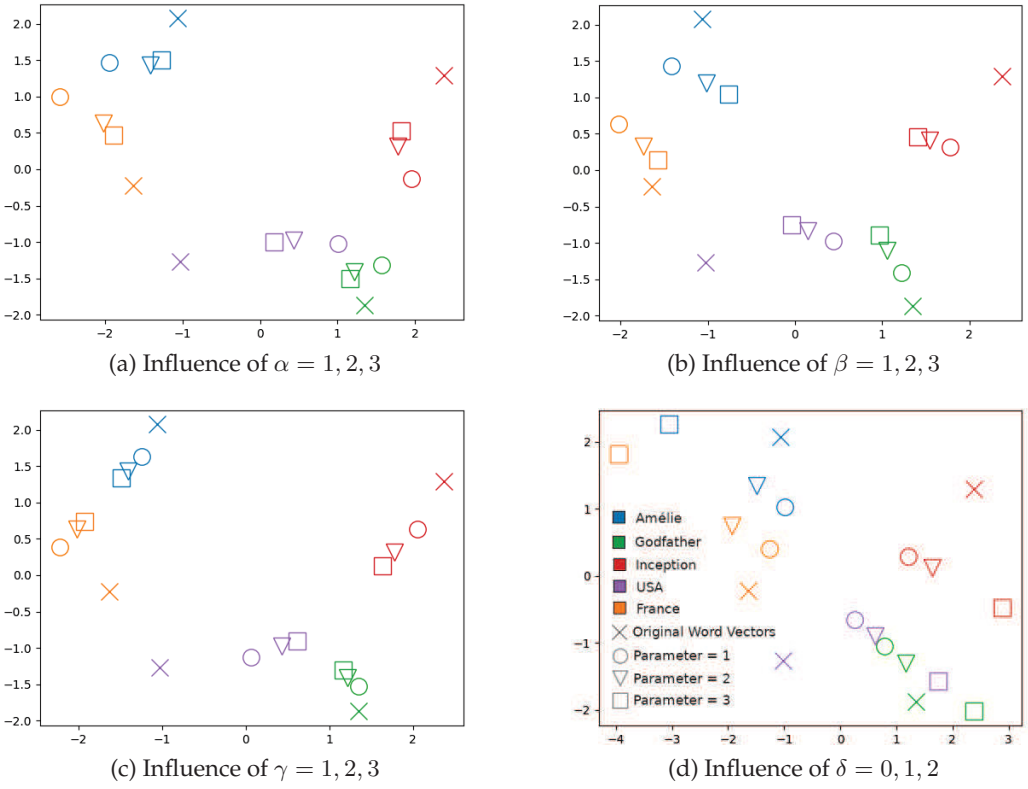
The retrofitting algorithm can be expressed as a set of matrix operations that can be solved efficiently. We update all vectors at once using a recursive matrix equation in a similar way as this was done in the retrofitting variant of [SC16]. $\Psi(W)$ can be minimized by iteratively calculating W^k starting with the base vectors W^0:

$$\begin{aligned}
W_R &= \sum_{r \in R} \Big[((\gamma_{ij}^r) + (\gamma_{ij}^{\bar{r}})^T) - ((\delta_{ij}^r) + (\delta_{ij}^{\bar{r}})^T) \Big] W^k \\
W' &= \alpha W^0 + \beta \boldsymbol{c} + W_R \\
D &= diag\Big(\alpha + \beta + \sum_{r \in R} \Big[\sum\limits_{\substack{j:(i,j) \\ \in E_r}} (\gamma_i^r + \gamma_j^{\bar{r}}) - \sum\limits_{\substack{k:(i,k) \\ \in \widetilde{E_r}}} (\delta_i^r + \delta_k^{\bar{r}}) \Big] \Big) \\
W^{k+1} &= D^{-1} W' \\
\boldsymbol{c} &= (\boldsymbol{c_1}, \ldots, \boldsymbol{c_n}) \quad \alpha = (\alpha_1, \ldots, \alpha_n) \quad \beta = (\beta_1, \ldots, \beta_n)
\end{aligned} \tag{5.14}$$

The matrices (γ_{ij}^r) and (δ_{ij}^r) are derived from γ_i and δ_i as defined in Section 5.2.3. Usually, a low number of iterations is sufficient. For our experiments, we set it to a fixed number of 10.

Parameter Configuration To configure the hyperparameters, we define four global parameters α, β, γ, and δ, from which we derive the setting of all other parameters. All α_i values are equivalent to the global value α. The other parameters are set to values that depend on the dataset. Every embedding $\boldsymbol{v_i}$ has one columnar connection and $|R_i|$ relational connections. Accordingly, we set the β_i values by taking into account the number of relationship types $|R_i|+1$ (including the columnar relationship) for weighting the influence of the columnar information. Respectively, the values γ_i are weighted by $|R_i|+1$ and the relation-group-specific outdegree values $od_r(i)$:

$$\beta_i = \frac{\beta}{|R_i|+1} \quad \gamma_{ij}^r = \begin{cases} \gamma_i^r = \gamma / (od_r(i) \cdot (|R_i|+1)) & (i,j) \in E_r \\ 0 & otherwise \end{cases}$$
$$od_r(i) = |\{j : (i,j) \in E_r\}| \tag{5.15}$$



(a) Influence of $\alpha = 1, 2, 3$

(b) Influence of $\beta = 1, 2, 3$

(c) Influence of $\gamma = 1, 2, 3$

(d) Influence of $\delta = 0, 1, 2$

Figure 5.4: Examples for Different Hyperparameter Settings

To fulfill condition (5.12), we set the δ values dependent on the maximal number of relation types mr and the maximal number of relations mc of any node in r.

$$\begin{aligned} \delta_{ij}^r = & \begin{cases} \delta_i^r = \frac{\delta}{mc(r)mr(r)} & (i,j) \in \widetilde{E_r} \\ 0 & otherwise \end{cases} \\ mr(r) = & max(\{|R_i| + 1 | (i,j) \in E_r \cup E_{\bar{r}}\}) \\ mc(r) = & max(|\{i : (i,j) \in E_r\}|, |\{j : (i,j) \in E_r\}|) \end{aligned} \tag{5.16}$$

Example. The influence of the hyperparameters is visualized in Figure 5.4: We retrofit 2-dimensional embeddings for a small example dataset containing three movies and the country where those movies have been produced. The base embedding model is obtained by projecting the popular Google News word2vec dataset (W2V-GN in Table 3.1) with PCA on two dimensions. There are two columns (movie and country) and one relation group (see Section 5.2.1). "Amélie" was produced in "France", the other movies in the "USA". We set the hyperparameters α, β, γ, and δ to different values and performed the relational retrofitting.

As shown in Figure 5.4a, the learned embeddings stay closer to their original embeddings when the α values increase. Higher values of β make it easier to cluster the columns from each other, e.g., reduce the distances between the movie vectors of "Inception" (red), "Godfather" (green), and "Amélie" (blue). The γ value controls the influence of relational connections. This brings the representations of text values that share a relation closer together. The δ factor causes vectors with different relations to separate and thus prevent concentrated hubs of vectors with different semantic. One can see in Figure 5.4d how $\delta = 0$ causes all vectors to concentrate around the origin of the coordinate system. If δ is set to a high value like $\delta = \alpha = 2$, the algorithm places the vectors far from the

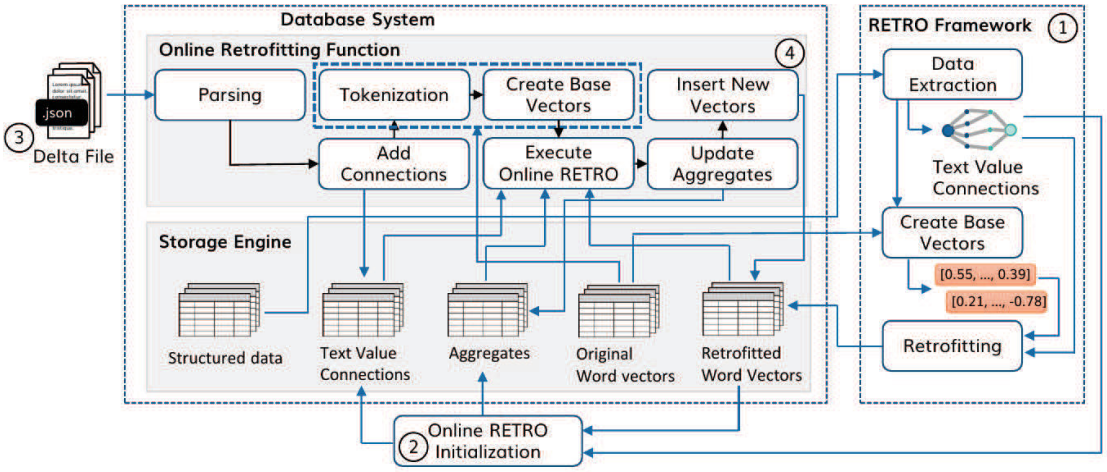


Figure 5.5: Online Retrofitting Process

origin of the coordinate system. However, related text values still get assigned to similar representations. In the example, the retrofitting algorithm is still converging for this configuration. However, for higher values of δ, some vectors will drift away more and more with every iteration.

Performance Optimization The maximal number of relations in E_r is defined by $|C_s| \cdot |C_t|$, where $|C_s|$ and $|C_t|$ refer to the cardinalities of the columns involved in r. However, in practice, most relations are much smaller, hence $|\widetilde{E_r}| >> |E_r|$. To optimize the extensive calculation of $((\delta_{ij}^r) + (\delta_{ij}^{\bar{r}})^T)W^k$ in (5.14), we utilize the following condition:

$$(((\delta_{ij}^r) + (\delta_{ij}^{\bar{r}})^T)W^k)_{i,*} = 2\Big(\hat{\boldsymbol{\delta}}^{\boldsymbol{r}} W^k - \hat{\delta}_i^r \sum_{j:(i,j)\in E_r} \boldsymbol{v_j}\Big)$$

$$\hat{\boldsymbol{\delta}}^{\boldsymbol{r}} = (\hat{\delta}_1^r, \ldots, \hat{\delta}_n^r) \quad \hat{\delta}_i^r = \begin{cases} \frac{1}{mc(r)\cdot mr(r)} & \exists k : (i,k) \in E_r \\ 0 & otherwise \end{cases} \tag{5.17}$$

By calculating the vector $\hat{\boldsymbol{\delta}}^{\boldsymbol{r}} W^k$ once and subtracting the centroid of the small number of related embeddings of every embedding $\boldsymbol{v_i}$, we can speed up the calculation for most cases.

5.2.4 Online-RETRO

When inserts and updates are performed frequently on a database, re-running RETRO would be too costly. Therefore, we developed an online update method to perform relational retrofitting only on a given set of text values, which can be hundred times faster than a complete re-run. Its implementation is integrated as a user-defined function into our PostgreSQL extension FREDDY.

An overview of the online retrofitting process is given in Figure 5.5. Before running the online retrofitting algorithm, it is necessary to execute the relational retrofitting algorithm (1) at least one time. Then, (2) initializes the database for the online retrofitting algorithm. Thereby, tables are created in the database, which contain information about

the connections between the text values, which have been extracted by the RETRO framework before. Moreover, aggregated values which are necessary for the online retrofitting algorithm are pre-computed. This involves $mc(r)$ and $mr(r)$ from Equation (5.16), a centroid for each column in the database to efficiently determine $\mathbf{c}$ (see Equation (5.14) and Equation (5.10)), and aggregates of word embeddings to perform a fast calculation of Equation (5.17).

To execute an online update, the database client needs to provide a delta file (3) with information about the changes to the database. This file contains a list of text values for which new embeddings should be calculated and their relations to other text values. For each of those text values and relations, an additional property should specify if it is added to the database since the last (online) retrofitting step. This information is necessary to update the information about text value connections in the first step of the online retrofitting algorithm (4). After that, the text values specified in the delta file are tokenized, and their base embeddings are calculated by using the word embedding model stored in the database. Then, those vectors are adjusted to the structured data in the database by the core online retrofitting algorithm. This algorithm works similarly to the retrofitting algorithm in RETRO. It consecutively adjusts the embedding vectors of the text values specified in the delta file, however, does not update all the other embedding vectors. Therefore, it uses the information about text value connections and the aggregated values from the database. Afterward, the aggregates of columns of text values in the delta file and the table with the retrofitted embedding representations are updated.

5.3 EVALUATION PLATFORM: RETRO LIVE

For studying the impact of the relational retrofitting process, we developed a Web-based evaluation tool RETROLIVE. The application has been presented at EDBT 2020, and a description of the evaluation platform has been published in [GTNL20].

5.3.1 Functionality

The tool guides the user through the whole retrofitting process. We integrated several database schemas and target word embedding datasets the users can choose from into the platform. RETROLIVE allows exploring their data statistics and characteristics. Moreover, it enables the user to set and fine-tune the various hyperparameters of the retrofitting learning problem. The impact of the hyperparameter values, the input data characteristic, and target embeddings can be studied in detail using either visualizations like 2-dimensional projections or by using word similarity and analogy benchmarks. To demonstrate the usefulness of the embeddings generated by the relational retrofitting approach, RETROLIVE comes with the implementation of several pre-defined classification and regression tasks for extrinsic evaluation. Furthermore, the users can define their own machine learning tasks. Finally, the platform also provides embedding representations generated with approaches such as the node embedding technique DeepWalk [PARS14] and the original retrofitting method [FDJ+15] from Faruqui et al. The user can use those as baseline embedding representations and apply them in the machine learning tasks via the RETROLIVE tool.

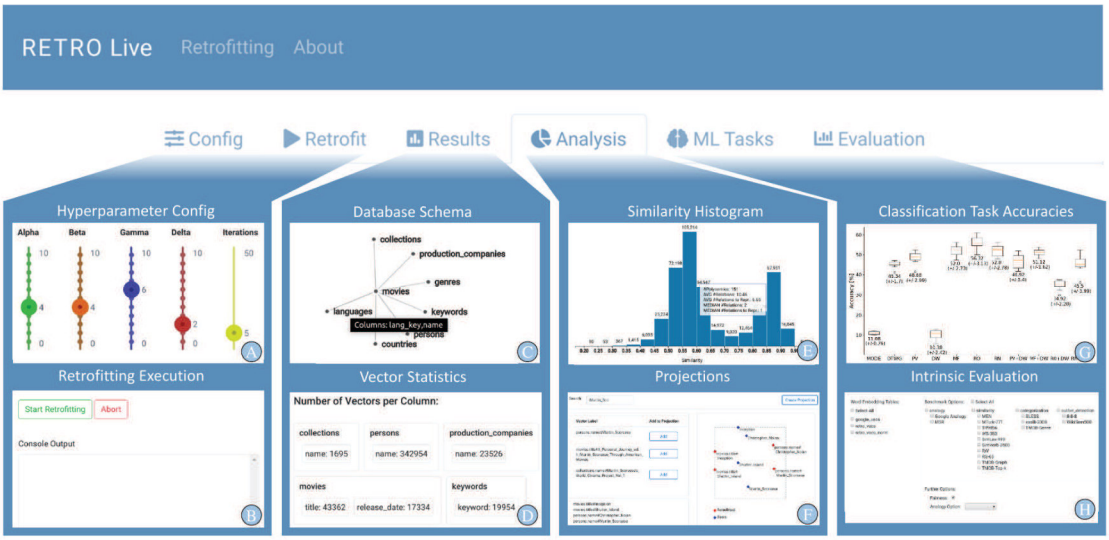


Figure 5.6: User Interface of RETROLIVE

5.3.2 Interface

Users access RETROLIVE through an interactive Web interface shown in Figure 5.6, where they can configure and explore the whole relational retrofitting process. There are six main views:

Config and Retrofit Those views allow to select the database (three are pre-defined) and the target word embedding model and configure the retrofitting process by setting the hyperparameters (A). For example, the users can specify lists of specific relations and columns for retrofitting to ignore in the config view. This is especially important if the retrofitting is executed on the same data used for the ML task, e.g., predicting the genres of movies expressed in the database by foreign key relations. Further, the relational retrofitting process can be triggered and monitored (B).

Results In this view, the user can inspect the extracted relational information (see Section 5.2.1) from the input schema in the form of graph (C). Additionally, the embedding statistics for each text column are presented (D).

Analysis In the analysis view, the users can inspect the characteristic of the retrofitted embeddings and compare them with the base embedding representations (plain). An interactive histogram (E) shows the distribution of the cosine similarity between the plain word vectors and the retrofitted vectors, i.e., to which degree certain vectors have been changed during the retrofitting process. The user can click on the individual bins to see additional information, e.g., in how many relations certain terms have been involved, in how many different columns a term appears, and a complete list of embeddings in the selected bin. For instance, one can observe that the average number of relations of a term

decreases with increasing similarity to the base embedding. Moreover, a 2-dimensional projection (PCA) shows the user-selected plain and retrofitted vectors (F). In the context of the TMDB example database[3], it can be seen that the vectors for movies and directors are arbitrarily distributed in the word embedding (red). However, after applying relational retrofitting, the movie and director vectors (blue) are clustered.

ML Tasks To show the benefits of relation retrofitting, the users can run different ML tasks (Section 5.4.4). The users select the embedding model (retrofitted, node, plain, etc.) they want to use for the given task. Training and testing data is retrieved from the database by using pre-defined SQL queries which can be also modified by the user. Diagrams visualize the results (G).

Evaluation We include 14 intrinsic evaluation tasks into RETROLIVE to test word similarity, e.g., SimLex999 [HRK15] MEN [BBBT12], or analogies, e.g., Google Analogy [MCCD13] (H). Here, the users can investigate whether original retrofitting and relational retrofitting affect the intrinsic task performance of the embedding representations.

5.4 EVALUATION

For the evaluation, we apply RETRO and the original retrofitting method on two popular datasets (see Section 5.4.1). For the base of the retrofitting, we employ different pre-trained word embedding models. The details of the training are discussed in Section 5.4.2. To evaluate the effect of the relational retrofitting on the performance of the embeddings in machine learning tasks, we implement several machine learning models described in Section 5.4.3. Afterward, we apply those models to various tasks on the two datasets, analyze the results, and compare them to the results of baseline approaches in Section 5.4.4. In addition, we evaluate the run-time of the relational retrofitting algorithm in Section 5.4.5 and the online retrofitting algorithm in Section 5.4.6. An intrinsic evaluation of the effect of the retrofitting algorithm is later presented in Section 6.4.2.

5.4.1 Datasets

We choose two popular real-world datasets: The Movie Database (TMDB)[4] and the Google Play Store Apps (GPSA) dataset[5]. The TMDB dataset consists of three CSV files for movies, credits, and user ratings where n:m relations, e.g., movies-genre, are encoded by JSON-like objects in the cells of the values. GPSA contains two tables for apps and their reviews. Both datasets are imported into a PostgreSQL database system. For TMDB, a database with 15 tables and $493,751$ unique text values in total is constructed. The GPSA database contains only 7 tables with $27,571$ text values. In the case of GPSA, we removed duplicates and apps without reviews.

[3] see Section 5.4.1 for more information on this database
[4] https://www.kaggle.com/rounakbanik/the-movies-dataset (Access: 08/26/21)
[5] https://www.kaggle.com/lava18/google-play-store-apps/ (Access: 08/26/21)

5.4.2 Training of Embeddings

We built RETRO on top of PostgreSQL. Given an initial configuration including the connection information for a database and the hyperparameter configuration, RETRO fully automatically learns the retrofitted embeddings and adds them to the given database. We trained relational embeddings for both databases, TMDB and GPSA, with a fixed number of 10 optimization iterations. For the binary classification and an imputation task, we used a grid search to identify good hyperparameter configurations for the relational retrofitting (RR) described in detail in Section 5.4.4. For the other ML tasks, we chose a configuration of $\alpha = 1, \beta = 0, \gamma = 3, \delta = 3$. We also trained word embeddings with the basic retrofitting approach of Faruqui et al., denoted as MF. Here, we use 20 iterations and the standard parameter configuration of $\alpha_i = 1$ and the reciprocal of the outdegree of i for β_i. We used two initial embeddings: The first one is obtained from the popular 300-dimensional Google News embeddings[6] (W2V) and the tokenization process from Section 5.2.1. The second one (SBert) is generated by the SentenceBert model bert-large-nli-mean-tokens[7] which is fine-tuned for the natural language inference (NLI) task. Node embeddings (DW) are trained with DeepWalk with its standard parameters and a representation size of 300.

5.4.3 Machine Learning Models

The embeddings provided by RETRO can be used as input for ML models to solve specific tasks. Here, we distinguish two cases: models that only use the RETRO embeddings and models that additionally incorporate the graph representation of the relations either by incorporating node embeddings or by using graph convolutional neural network layers that can directly operate on the graph whose nodes we annotate with the embeddings generated by RETRO. While RETRO already uses relational information implicitly, we argue that ML models can still profit from their explicit encoding.

Node Embeddings

Node embedding techniques, which we described in Section 2.2.6, are frequently used to create vector representations for nodes in graphs to apply ML models to them. Most of these embedding techniques try to capture certain properties of the neighborhood of nodes in spatial relations of the node vectors. In this work, we use the node embedding technique DeepWalk [PARS14] that has already been applied successfully for data integration tasks [IU18]. Given the graph representation of the database text values (see Section 5.2.1), node embedding techniques can be directly applied without any additional effort. For our work, we use the embeddings generated by DeepWalk in two ways: first, as a strong baseline in our evaluation (see Section 5.4.4). Second, we follow the notion of [GAS16] that showed that combining word embeddings with node embeddings by simple vector combination methods improves their performance in intrinsic word similarity tasks. Specifically, the authors showed that a combination of word embeddings and node embeddings captures the human notion of similarity more accurately than word embeddings themselves. During testing several combination methods, we decided to use the concatenations of both embeddings since those show good improvements across different ML tasks.

[6] https://code.google.com/archive/p/word2vec/ (Access: 08/26/21), W2V-GN in Table 3.1 in Section 3.2
[7] https://github.com/UKPLab/sentence-transformers (Access: 08/26/21)

Artificial Neural Networks

Artificial neural networks provide general-purpose solutions for ML models operating on textual and relational data. In this work, we provide models applicable to common classification tasks used on database systems which are: *binary classification*, *multi-class classification*, e.g., missing value imputation, and *link prediction*. For the binary classification, a classifier has to determine if a text value is assigned to a label or not. In case of a category imputation problem, the classifier can select a label from a set of labels. The link prediction problem is typically defined on graphs where the goal is to predict links that are missing or likely to be created in the future like a probable friendship relation in a social network [LNK07]. In our case, we consider the link prediction task for a specific relation. We train our embeddings without considering the respective relations. Afterward, we take a portion of word pairs that are linked by the relation and a portion of words where no relation exists and train a neural network to predict whether the relation is present. This is a similar procedure as done in [LMP18].

In order to learn from word embedding inputs, rather simple (deep) feed-forward networks are commonly used [GFEC16] as well as complex models like graph neural networks [PLH+18]. In this work, we implement simple networks which typically need less run-time and complex models with generally higher classification performance.

Feed-forward Networks: For the multi-class classification, we create an ANN with two hidden layers. The first layer has 600 and the second 300 neurons. We use sigmoid activation functions for the fully connected inner layers. For binary classification, one 300-dimensional hidden layer is sufficient. In the case of the binary classification, a sigmoid, and in the case of category imputation, a softmax function is used for the output layer. We use binary cross-entropy as a loss function for binary classification and categorical cross-entropy for category imputation.

For link prediction, we use an ANN, which gets an edge encoded by a source and a target embedding. Both embedding layers are processed by an inner layer, then get subtracted, and the result is processed by another layer, which is then connected to the output. Sigmoid is used as an activation function and binary cross-entropy for the loss. The binary output value classifies the edge as present or not.

To prevent overfitting, we added dropouts [SHK+14] and L2 regularization for the binary classification.

Graph Convolutional Neural Networks: One class of graph neural network approaches (see Section 2.2.6) is the class of Graph Convolutional Neural Networks (GCNs) [KW17, HYL17]. Those are widely applied for node labeling but have also been used for text classification [PLH+18, YML19]. There are several variants of GCN approaches. The design of our GCN model is inspired by GraphSage [HYL17]. Here, the input of a GCN is a graph $G = (V, E)$ with features $X \in \mathbb{R}^{n \times m}$ annotated to the nodes V, where n is the number of nodes and m denotes the dimensionality of the input feature vectors. In the training process, the network learns to predict labels for nodes in a graph, given an incomplete set of labels as the training set. In this way, we can use GCNs to label text values in a database with the graph definition from Section 5.2.1. In detail, the GCN is defined by a list of hidden layers $H_i^{m'' \times m'}$ and a message-passing function $\mu : V \to \mathbb{R}^{m'}$ which accumulates for a node the features assigned to it and the features of its neighboring nodes $N_v \subseteq V$ to a single vector. To calculate the output X^{l+1} of each layer H_l,

μ accumulates the accumulated feature vectors, a linear projection with the matrix H_l is applied, and the $ReLU$ activation function processes the result:

$$X_v^{l+1} = ReLU(H_l \cdot \mu(v)) \tag{5.18}$$

The output X_v^{l+1} forms the node's features applied to the following layer. The last layer represents the labeling of the nodes. Our message-passing function averages the features of neighboring nodes and concatenates the result with the features of the node itself:

$$\mu(v; N_v, X^l) = \Big(\frac{1}{|N|} \sum_{i \in N_v} X_i^l\Big), X_v^l \tag{5.19}$$

This allows the network to separate between features of the node itself and the accumulated features from neighboring nodes in the first layer. In our experiments, we use only one 300-dimensional hidden layer and a layer to predict the labels. GCNs scale linearly with the number of nodes in the graph. In our use case, this corresponds to the number of text values in the database. Therefore, we suggest reducing the training run-time for large databases by using neighbor sampling [HYL17]. This is done by selecting only up to a certain number s of neighbors $N_v' \subseteq N_v$ to calculate the aggregated value $\mu(v; N_v', X^l)$ of a node v for each layer.

Training: All networks are trained with the NAdam optimizer [Doz16]. Following common practices to prevent irregular updates of the weights, we normalize the embedding vectors before they are processed by the network. During training, we monitor the loss on a validation set, stop training if it does not improve for 50 epochs, and select the model with the lowest validation loss. The accuracy is determined on a separate test set.

5.4.4 Evaluation of ML Models

Competitors

To demonstrate the applicability of RETRO and its learned representations for different tasks, we use the following baseline approaches:

DataWig: Missing value imputation on CSV files with text values can be accomplished with DataWig [BSS+18]. This recently published category imputer is based on n-gram representations of text values that are utilized by LSTM neural networks to assign single text values or rows of text values in a CSV file to categories. As an input, the imputer gets a CSV file, a list of columns used for the imputation, a column, which contains the values which should be assigned to categories, and the column which usually holds the output category. The imputation is then trained on a sample set of rows. We compare DataWig against imputation using our embeddings in Section 5.4.4.

Mode Imputation: A very simple imputation method is to replace a null value with the mode value (most frequent value) in the column. According to [BSS+18], most data-wrangling frameworks implement only such simple imputation methods for category imputation, especially if there is only non-numerical data given. Mode is a very popular and often used imputation method and, thus, it also serves as a good baseline.

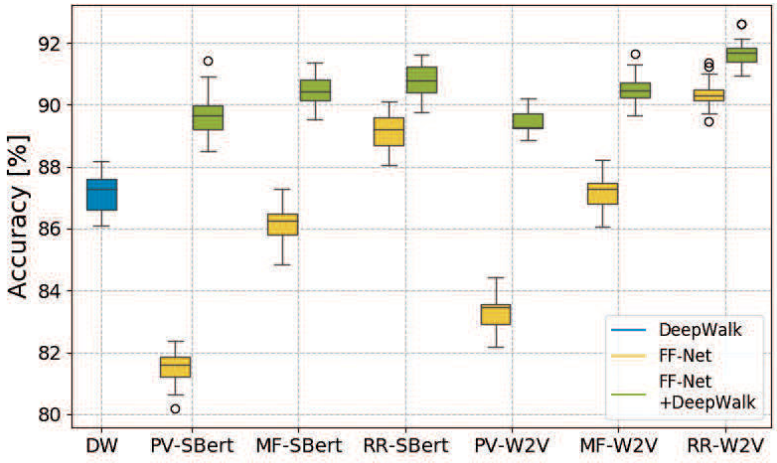


Figure 5.7: Binary Classification of US-American Directors with Different Embedding Types

DeepWalk: Node embeddings generated by DeepWalk (see Section 5.4.3) are often used for link prediction tasks and thus can be used to predict missing foreign key relations in database systems. The prediction is usually performed using feed-forward neural networks similar to the use of text value embeddings proposed in this paper. The details are described in Section 5.4.4. DeepWalk not only serves as a baseline but can also be combined with other text embeddings by concatenation [GAS16].

Binary Classification

We use the feed-forward network architecture described above to classify directors of the TMDB dataset as either US-American or non-US-American. We extract the citizenship from Wikidata [VK14] by using the SPARQL query service. Thereby, we derive $33,647$ directors holding in total $37,203$ citizenships. We omit 387 directors holding the US-American and another citizenship according to Wikidata because this could be considered ambiguous. From the remaining set of directors, the TMDB database contains $9,054$ persons, which are considered for the classification.

For all experiments, we sample 10 times embeddings of $3,000$ US-American directors and $3,000$ non-US-American. One half of a sample set is used to train the ANN. The other half is used as a test set. The retrofitting approaches are performed on the Google News (W2V) and SBert embeddings. For the Google News (W2V), we executed the classification on different hyperparameter settings to investigate their influence on this task. Figure B.1a in Appendix B visualize the average accuracy values for all tested parameters. One can see that high values for γ and δ deliver good results. This suggests that relational information is more important for a good classification in this task. Configurations with a high value of δ but low values for α and γ lead to non-converging configurations and worse classification results. The classification is also performed on embeddings derived by concatenation of DeepWalk embeddings and the other embedding types. In combination with node embeddings (Figure B.1b), the optimal configuration has higher values for α and β, since the relational connections are already represented by the node embeddings. The β parameter has a low influence on this task since all inputs are part of the same column. Figure 5.7 shows the distributions of the recognition accuracy values achieved by the classifier for RR ($\alpha = 1, \beta = 0, \gamma = 3, \delta = 3$) and the other embedding

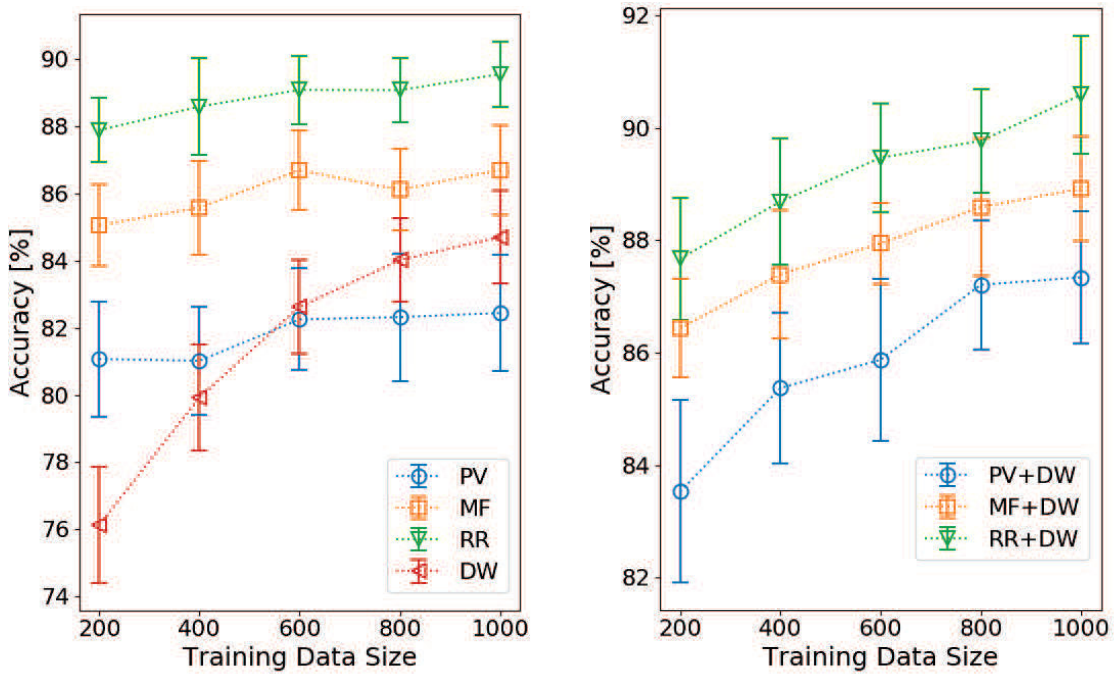


Figure 5.8: Classification of Birth Places of US-American Directors with Increasing Sample Size

types. Relational retrofitting (RR) achieves the best accuracy values on both word embedding datasets. W2V-based embeddings deliver slightly better results, however, this does not hold for all ML tasks, as shown by the results below. The node embeddings (DW) are outperformed by all other types, besides the plain word embeddings (PV) and the baseline retrofitting approach (MF), which achieves similar performance. However, combining node embeddings with all other approaches increases the achieved accuracies above 90% in all cases except the plain word vectors. Additionally, we ran the experiments for varying numbers of training samples for W2V. We trained the neural network with 200 to $1,000$ samples and validate the accuracy with $1,000$ test samples. Since the deviation is much higher for such small sample sets, we trained the ANNs 20 times. The result is shown in Figure 5.8. The influence of the training sample size is at the lowest for the plain word embeddings. If the sample sizes are small, DeepWalk is getting outperformed by the plain word embeddings. Hence, DeepWalk needs a larger amount of training data to achieve comparable results.

Missing Value Imputation

As a basic data integration task, we perform missing value imputation for categories on both datasets using the feed-forward and GCN architectures described for multi-class prediction in Section 5.4.3.

Imputation of Movie Languages: In the TMDB dataset, "original language" is an attribute of movies with exactly one value for each movie. Subsequently, the prediction of those attribute values can be considered as a typical value imputation problem. To perform the classification, we ignore the "original language" column in the movie table

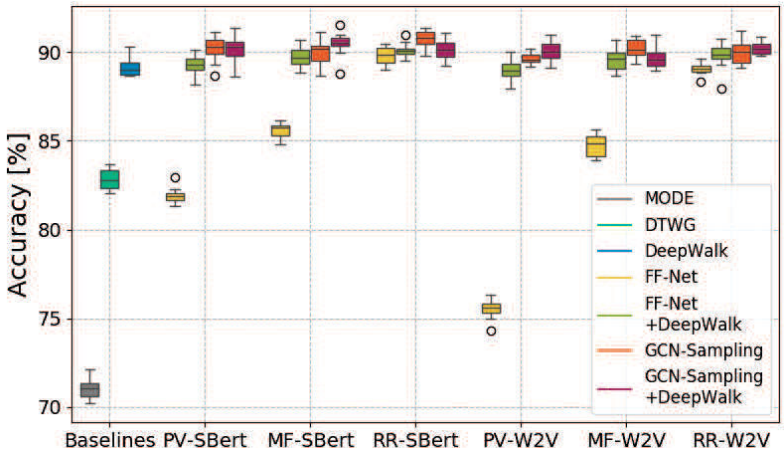


Figure 5.9: Comparison of Imputation Methods for the Original Language Attribute

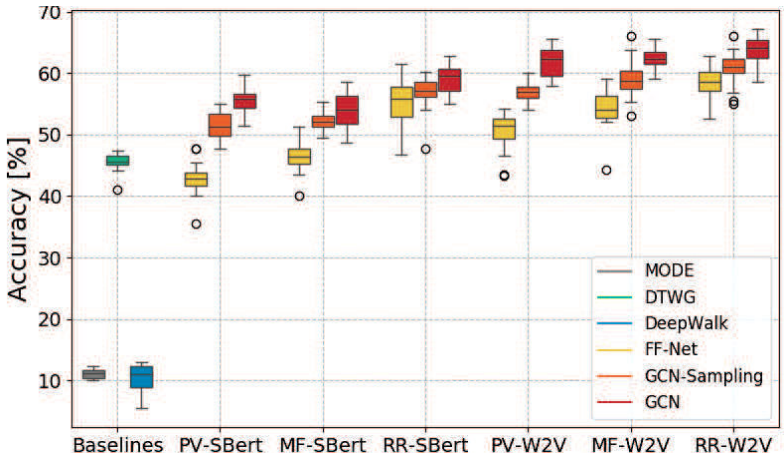


Figure 5.10: Comparison of Imputation Methods for App Categories

while retrofitting the embeddings. Afterward, we train the networks using $5,000$ training samples and $5,000$ samples to evaluate the accuracy. Sampling, training, and evaluation are repeated 10 times for each embedding and network type. In addition to the embedding approaches, we apply mode imputation and DataWig with an equivalent sampling strategy. DataWig is provided with textual movie information in the form of the original CSV file. It contains all information imported into the database, except directors and actors, which reside in other tables. Figure 5.9 compares the accuracy values for all methods and embedding types. Since most of the movies are in the English language, the mode imputation (MODE) performs quite well and achieves an accuracy of 71.09% in the average case. For this task, SBert-based embeddings perform better in comparison to W2V-based ones. Using the feed-forward architecture DeepWalk achieves good accuracy values, which are clearly better than the DataWig results and only slightly lower than the results of the relational retrofitted embeddings ($\alpha = 1, \beta = 0, \gamma = 3, \delta = 3$). This suggests that relational information is important for this task, which may be the case since the original language can often be derived from the set of languages spoken in the movie stored in the database. Word embedding information can only contribute a little to better classification. In combination with DeepWalk, all embedding types improve on the task. We also tested the GCN network, however, only in combination with the neighbor sampling because the size of the database leads to high resource consumption otherwise. GCNs achieve the highest average accuracy in this setting. Because of the high prevalence of relational features, no clear difference is recognizable between the types of text embeddings used in the network.

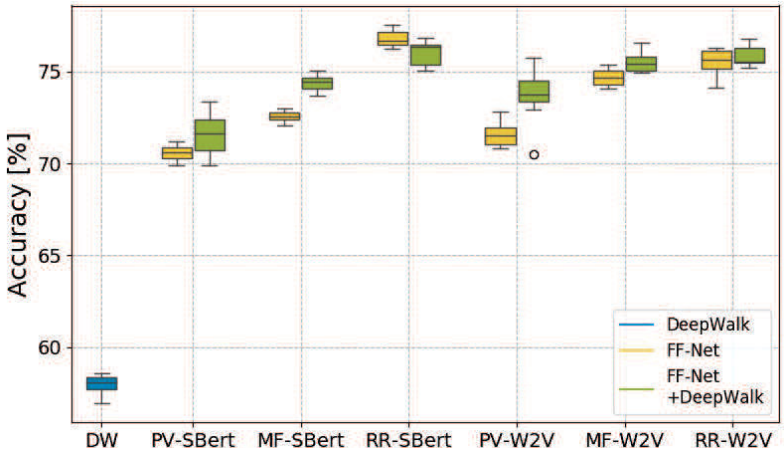


Figure 5.11: Link Prediction for Genres

Imputation of App Categories: In the GPSA dataset, we classify the category attribute for each app. For the training of the embeddings, we omit the category information and the genre relation since genre and category are often equivalent. The apps in the dataset are grouped into 33 categories. Again, we compared our imputation to DataWig and the mode imputation. Since DataWig can only be executed on singular tables, we omit the review data. We sampled 10 times two disjunct sets of 400 apps as training and testing sets. The resulting accuracy values of the imputation are shown in Figure 5.10. Here, the mode imputation achieves only very poor accuracy values since the apps are distributed more evenly over the category values. DataWig achieves clearly better performance, which is similar to the performance of the plain word embeddings of the app name. This is probably the case because DataWig might also rely strongly on the app name since all other attributes are quite unrelated to the category information. The relational retrofitted embeddings ($\alpha = 1, \beta = 0, \gamma = 3, \delta = 3$) can utilize the reviews and achieve an up to 13% more accurate result only with the feed-forward architecture. DeepWalk only reaches a classification accuracy, which is comparable to mode imputation, since the classification can not be done based on relational information. Accordingly, a concatenation with other embeddings does not make sense for this task. When using the GCN architecture on the embeddings, the accuracy improves on all settings. Also in this setting, the relational retrofitted embeddings achieve the best results. As one would expect, GCNs without neighbor sampling achieve better results compared to GCNs with sampling.

Link Prediction

To evaluate our model for the link prediction, we decided to predict the movie-genre relations in TMDB. There are 20 genres in total. Usually, a movie is assigned to multiple genres in the dataset. This prediction is difficult because the metadata present in the dataset itself provides only limited information about the genres. Thus, the classification should take into account knowledge encoded during the pre-training of the word embedding models. Another difficulty constitutes the fact that it is quite subjective in which genre a movie fits, and it can be assumed that the genre information is incomplete since some movies are not assigned to a genre. Furthermore, the genre information is rather diverse. For example, one genre is "TV Movie", which refers to the media it is published in, while the majority of the genres refer to content aspects like this is the case for "Horror" or "Comedy". For training our neural network, we select sets of $5,000$ movie-genre relations and $5,000$ arbitrary connections between movies and genres which are not in the dataset and serve as negative examples. Then, we split those pairs into $9,000$ samples for

TMDB	**MF**	**DW**	**RR**
Run-time	$7.39s$	$548.72s$	$418.13s$
Deviation	$\pm 0.07s$	$\pm 0.83s$	$\pm 1.15s$
GPSA	**MF**	**DW**	**RR**
Run-time	$12.23s$	$1130.63s$	$178.78s$
Deviation	$\pm 0.30s$	$\pm 13.85s$	$\pm 0.55s$

Table 5.1: Run-time of Embedding Methods

training and $1,000$ samples for testing. We repeated this 10 times. Figure 5.11 presents the results of the accuracy evaluation. The DeepWalk (DW) embeddings usually used for link prediction fail in this setting. This is probably the case because the node vector of the genres are potentially meaningless since all of those nodes have only a single edge to the same blank node. Retrofitted vectors clearly outperform the plain word embeddings where relational retrofitting (RR) is superior to standard retrofitting (MF). In combination with node embeddings, some text-based approaches achieve better results.

5.4.5 Run-time Measurements

We measure the training time of the two retrofitting methods and DeepWalk by measuring their execution when executed in a single thread for both datasets. The measurements are repeated 10 times. Due to the very high training times of DeepWalk, we used a subset of the TMDB database with 12,593 unique text values. The results of the measurements are shown in Table 5.1. As one can see, the retrofitting algorithms (MF and RR) are faster than the DeepWalk (DW) node embedding method. The fastest method is the basic retrofitting approach (MF) which is expected since relations are modeled in a much simpler fashion. This leads to lower accuracy in ML tasks, as shown in the evaluation results above. Comparing the run-times between GPSA and TMDB, we see an increase for all methods except relational retrofitting (RR). This behavior is explained by the larger amount of relational groups in TMDB compared to GPSA, leading to a larger amount of comprehensive matrix multiplications, which is the most time-consuming routine for relational retrofitting.

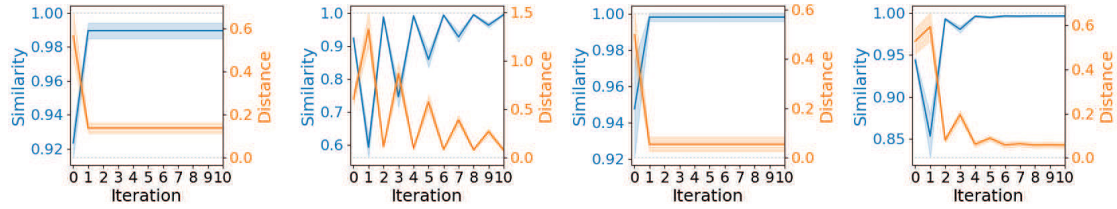
5.4.6 Online Retrofitting

To evaluate the online retrofitting algorithm, we designed two user scenarios for the data in the GPSA dataset. For both scenarios, we measure the run-time and the accuracy of the algorithm.

Setup of Experiments: To evaluate the online retrofitting, we apply a sampling function to randomly remove text values, run the retrofitting on the reduced dataset, and online update the missing values. However, completely random sampling would lead to unrealistic setups. Therefore, we designed two specific sampling functions for the GPSA dataset modeling two more realistic scenarios: The first sampling function samples 10 apps for which we only removed one randomly selected review each. In contrast, the second sampling function also samples 10 apps but removes the apps together with all associated reviews. This usually involves multiple hundred text values. In the evaluation tasks below, the experiments are executed 10 with different sets of text values generated by the respective sampling function. As base word embedding model, the W2V-GN model is used.

Process	Only Reviews	Apps and Reviews
Online-RETRO Initialization	$5.448s$ ($\pm$ 0.087)	$5.482s$ ($\pm$ 0.083)
Parse Delta File	$0.001s$ ($\pm$ 0.000)	$0.022s$ ($\pm$ 0.004)
Add Connections	$0.001s$ ($\pm$ 0.001)	$0.016s$ ($\pm$ 0.003)
Retrieve Old Vectors	$0.041s$ ($\pm$ 0.013)	$0.039s$ ($\pm$ 0.012)
Create Base Vectors (incl. Tokenization)	$11.221s$ ($\pm$ 0.142)	$11.165s$ ($\pm$ 0.141)
Online Retrofitting	$0.019s$ ($\pm$ 0.007)	$0.423s$ ($\pm$ 0.066)
Update Aggregates	$0.001s$ ($\pm$ 0.000)	$0.005s$ ($\pm$ 0.000)
Insert Vectors	$0.003s$ ($\pm$ 0.000)	$0.071s$ ($\pm$ 0.012)

Table 5.2: Run-time of Online Retrofitting



(a) Only Reviews $\alpha = 1\ \beta = 0\ \gamma = 3\ \delta = 3$ (b) Apps and Reviews $\alpha = 1\ \beta = 0\ \gamma = 3\ \delta = 3$ (c) Only Reviews $\alpha = 1\ \beta = 1\ \gamma = 1\ \delta = 1$ (d) Apps and Reviews $\alpha = 1\ \beta = 1\ \gamma = 1\ \delta = 1$

Figure 5.12: Online Retrofitting Evaluation

Run-time Evaluation: To analyze the run-time properties of the online retrofitting, we divide it into seven sub-procedures (see Figure 5.5) and measure their execution time in the two scenarios. In addition, we measure the execution time of the initialization of Online-RETRO. Table 5.2 shows the mean run-time values and the standard deviation. One can see that the overall execution time is much lower than the execution time of a complete re-run of the retrofitting (see Table 5.1). The most time-consuming sub-procedure is the creation of the base vectors. It requires even more execution time than the initialization. This is the case since for the tokenization of the text values, the algorithm builds a trie of all tokens in the vocabulary of the word embedding model annotated with the embedding representations (see Section 5.2.1). This process could potentially be optimized by storing and indexing the trie in the database to determine the tokenization efficiently and only retrieving the required vectors. Besides this process, the retrieval of the old vector dataset and the Online-Retro sub-procedure are the most time-consuming steps in the calculation. Here one can notice that the Online-Retro process requires much more time in the second scenario. The run-time of the remaining sub-procedures is relatively short.

Deviation from Re-running RETRO: The vectors resulting from the online retrofitting algorithm can slightly differ from the results of a re-run. This is specifically the case since the vectors generated by the retrofitting before the execution of the online retrofitting, which are not updated during Online-Retro, are learned without the knowledge of the later added text values. Embeddings added later then adapt to those slightly different embeddings and thus also obtain slightly different embedding representations. To quantify this delta, we calculate for both scenarios the average Euclidean distance and the cosine similarity of the resulting embeddings after each of 10 online retrofitting iterations to the results of a complete re-run. For both scenarios, we execute this evaluation process with the hyperparameter settings $\alpha = 1, \beta = 0, \gamma = 3, \delta = 3$ and $\alpha = 1, \beta = 1, \gamma = 1, \delta = 1$. Figure 5.12 visualizes the mean and the standard deviation in each iteration. If only

reviews are inserted, one online retrofitting iteration is enough to achieve 0.99 or higher cosine similarity (see Figure 5.12a and Figure 5.12c). In the following iterations, nothing changes because the reviews are not directly related. In the second scenario, however, there are relations between new text values, ie., app names and reviews, whose vectors get updated. In the first iteration, reviews get more similar to the original embeddings of app names, and the app name embeddings get more similar to the original embeddings of the reviews. In the next iteration, this reverts and leads to the alternating behavior shown in Figure 5.12b and Figure 5.12d. This leads to the highest similarity in iteration 10, where the embeddings are very similar to the embeddings determined by a re-run (see Figure 5.12b and Figure 5.12d). The remaining deviation results from short reviews like "Great" and "Nice", which are shared by multiple apps. In general, the deviation is higher with the hyperparameter setting $\alpha = 1, \beta = 0, \gamma = 3, \delta = 3$ (see Figure 5.12a and Figure 5.12b) since the values of the optimal embedding representations depend stronger on related text values in a setting with high values of γ and δ.

5.5 SUMMARY

In this chapter, we investigated methods to optimize embedding representations of text values in database systems. In Section 5.1, we surveyed approaches to utilize the structure knowledge from the database to adapt the embedding representations to the context-specific meaning of their text values. We concluded that the concept of retrofitting qualifies best for the requirements defined in Section 3.6. Accordingly, we decided to implement a novel relational retrofitting algorithm for text values residing in relational database systems. This led to the development of the framework RETRO (Section 5.2) for automatically generating optimized embeddings for text values in relational database systems. RETRO is complemented with Online-Retro to generate optimized representations for newly inserted text values without re-running RETRO itself (see Section 5.2.4). In addition, we implement the evaluation platform RETROLIVE to investigate the influence of hyperparameters and the resulting embeddings (see Section 5.3). To execute Online-Retro from the database, we integrated it as a UDF into the FREDDY Extension (see Section 4.1). Our Evaluation in Section 5.4 demonstrates that relational retrofitting is more efficient than the common node embedding techniques DeepWalk and thus can be used for optimizing text values in large datasets. We further show how ML models for several tasks can profit from relational retrofitting and achieve better results as achieved by the original retrofitting algorithm on two real-world datasets. To evaluate Online-Retro, we analyzed its execution on two user scenarios on real data in Section 5.4.6 In those settings, Online-Retro allows generating nearly optimal embeddings for inserted text values much faster than re-running the RETRO framework.

6

MODEL RECOMMENDATION

To select a suitable word embedding model for a query in a word embedding database system, a solution to recommend embedding models is necessary. Furthermore, the training of high-quality word embedding models requires large text corpora, e.g., 840 billion tokens for the popular pre-trained GloVe Common Crawl model (GV-CC in Table 3.1), which in many cases are not available. Thus, it is common to reuse pre-trained general-purpose models. Therefore, machine learning developers frequently face the problem of choosing a pre-trained embedding for their given task. For instance, for building a search engine for a movie website, they should be sure that the selected word embedding reflects the movie domain appropriately. Besides simply reusing pre-trained models, they also can be adapted to better fit a domain-specific context using retrofitting, as discussed in Chapter 5. Moreover, higher quality domain-specific word embeddings can be produced by mapping domain-specific embeddings in a general-purpose word embedding model, as shown in [SLS18]. In all those cases, identifying the best pre-trained word embedding for a given task is crucial. To solve this problem, we propose a framework to automatically construct an evaluation dataset for domain-specific evaluation. We employ this framework on a large corpus containing 125M tables extracted from HTML Web pages to construct a novel benchmarking dataset, which assists ML developers with selecting a suitable model. The results of our research on this topic are already published in [GSTL20b]. Parts of this chapter are based on this work. The dataset itself is also published in [GSTL20a].

In the following, we present our work on the extraction framework, which leads to the construction of our evaluation dataset FACETE. This dataset is designed to fulfill the requirements presented in Section 3.7. Specifically, it compromises a large number of relations, consists of continuous valid facts of the extended general knowledge, covers several domains, and is flexible enough to adapt an evaluation to the application domain of the word embedding models. To approach this, we first review existing methods for word embedding evaluation in Section 6.1. Then, we explain the desired architecture of our evaluation benchmark in Section 6.2. In the following, Section 6.3 describes the construction pipeline for FACETE. Afterward, we present the results of an exemplary evaluation of popular pre-trained embedding models using FACETE in Section 6.4 and finally conclude in Section 6.5.

6.1 RELATED WORK

As stated in Section 2.3, word embedding evaluation methods are categorized into extrinsic and intrinsic methods in the literature. In the following, we survey methods in both categories for domain-specific evaluation and discuss their limitations.

6.1.1 Extrinsic Evaluation

Extrinsic evaluation methods capture quality measures for results of an application task obtained by using different embedding models. For instance, one can evaluate the accuracy or the F1 measure of an ML algorithm which utilizes embedding representations provided by the given embedding models. Recently, several benchmarks to test language understanding and question answering have been proposed [RZLL16, ZBSC18, WSM+19]. Especially contextualized embedding model like BERT are evaluated by extrinsic evaluation on such tasks [DCLT19]. Because of this focus on specific applications, extrinsic methods are always domain-specific. An extrinsic evaluation task should be designed to be as close as possible to the actual application. In [RK19], one can find a good example for an extrinsic evaluation for static embedding models trained to capture the semantic of words in a specific domain. To obtain an accurate estimate of the performance of an embedding model in an application, extrinsic evaluation is preferred over intrinsic evaluation. However, extrinsic evaluation has also several limitations:

Limitations: [SLMJ15] shows that the correlation of the evaluation results across different extrinsic tasks is low. Thus, one should not deduct from the outcome of one extrinsic evaluation task the performance of a model in a different task. Therefore, extrinsic evaluation is only possible if a comprehensive gold standard for the application task is available. Especially for unsupervised tasks, this may constitute a problem. Moreover, intrinsic methods are preferred when an exhaustive extrinsic evaluation is too time-consuming. This particularly applies to the training of deep learning models, which are common in natural language processing, and require high computational effort. In contrast, the results of an intrinsic evaluation are directly available since they are independent of the application task.

6.1.2 Intrinsic Evaluation

Section 2.3 already provides a comprehensive overview of datasets for the intrinsic evaluation of word embeddings. Those datasets are designed by the NLP community to cover a wide range of linguistic relations in form of evaluation tasks that involve frequently used words. However, as far as we know, there is no dataset that provides an overview of the performance of a word embedding model according to different domains. Besides those generic datasets, there is a small number of benchmarks for very specific domains. For instance, in [NØL18], the "Schlumberger oilfield glossary" (slb)[1] was constructed from which the authors derive 878 synonym pairs, 284 antonym pairs, and 934 alternative form pairs used to intrinsically evaluate their model for the oil and gas domain. For applications designated for tasks involving text from other domains, only generic intrinsic evaluation datasets are applicable. However, here several problems exist:

Limitations: Intrinsic evaluation datasets proposed by the NLP community (see Table 2.1 in Section 2.3) facilitate researchers to compare embedding techniques trained on the same text against each other. However, for the application of pre-trained word embedding models in database systems, the goal is slightly different. Here, one aims to evaluate pre-trained models usually trained on different texts to represent semantic knowledge in the domain of the database. Most of the existing data collections are relatively small and map the word embedding model on just a single number, neglecting all

[1] http://www.glossary.oilfield.slb.com/ (Access: 08/03/21)

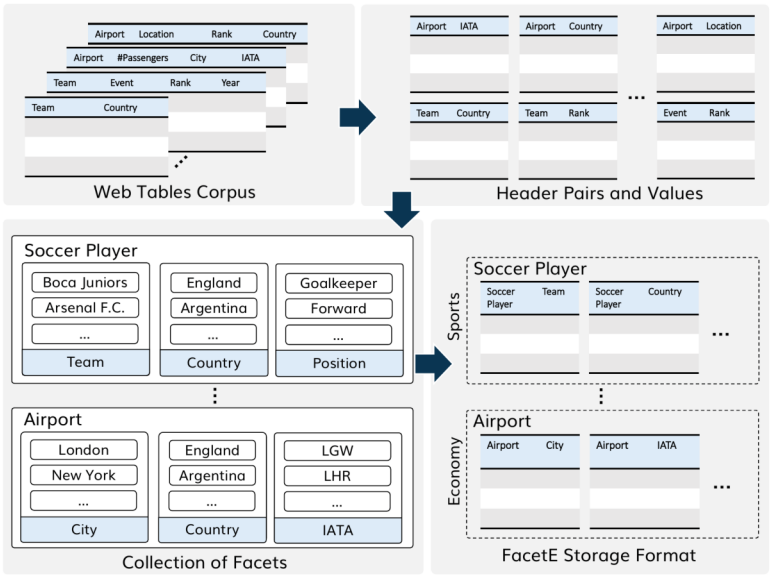


Figure 6.1: Facet Data Structures

variations of the performance in different domains. Moreover, many of them have been labeled manually and therefore are biased by certain factors, e.g., the subjectiveness of rating scales and the meaning of similarity, as well as the lack of penalties for overestimating similarity of two dissimilar words [BKR+16, AG16]. Some of the datasets contain a few categories of relations (e.g. WordRep [GBL14]). However, they are too generic and primarily consist of linguistic categories. Thus, a fine-granular evaluation according to Requirement D2 and the desired flexibility (Requirement D5) are not provided. Besides, an automatic adaptation for a specific domain is not possible (Requirement D6).

6.2 ARCHITECTURE OF FACETE

To allow the user flexibility in the evaluation of embedding models according to Requirement D5, we employ the notion of facets to structure our dataset.

Facet-Oriented Data Structure The dataset should be structured in multiple so-called facets, which are categorized in a handful of broader categories as displayed in the bottom right-hand corner in Figure 6.1. Facets are frequently used for classification purposes where one or multiple facets are applied to classify objects to category values or combinations of values. Facet classification originates from systems to organize books in libraries [Ran39]. Later it has also been adapted for information retrieval systems [PD91, BL00]. In contrast to hierarchical classification schemes, multiple facets can be applied to one object. For a particular domain, only certain facets are relevant. A facet classification scheme is flexible in the sense that it allows to select only those facets relevant for a specific use case or a domain and do the classification based on them [Her07]. This makes the concept of facets appealing for the data structure of our evaluation dataset. For the storage schema, we choose to represent every facet as an injective relation $F \subseteq O \times V$, which is stored as a set of tuples.

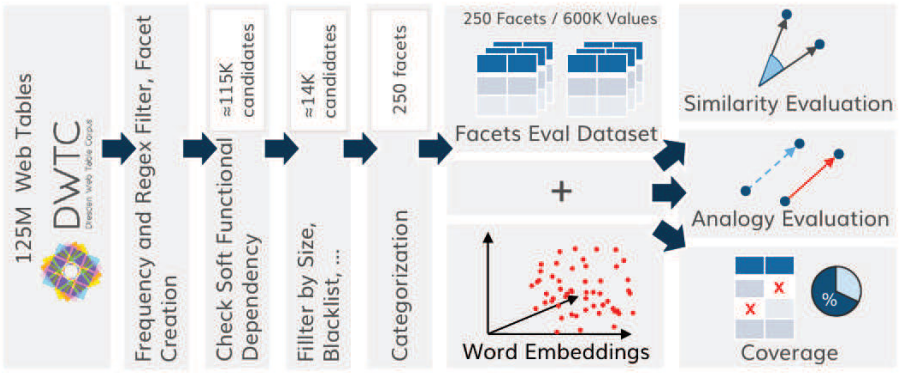


Figure 6.2: System Overview: Extraction and Evaluation

Example Given a specific object set O (e.g. "soccer players"), one can apply facets to it. Figure 6.1 shows in the bottom left-hand corner a set of facets for soccer players ("team", "country", "position"). Each facet holds a set of values V to which the objects (e.g. soccer players) can be assigned.

6.3 EVALUATION DATASET CONSTRUCTION PIPELINE

To compile an intrinsic evaluation dataset that fulfills the requirements mentioned in Section 3.7, we decide on a data-driven approach. Specifically, we leverage the Dresden Web Table Corpus (DWTC) [EBH+15] that consists of 125 million Web tables. The extraction process, shown in Figure 6.2, is divided into four steps: 1) Filtering Web tables and facet candidate generation, 2) determining soft functional dependencies to decide which facets and facet values are appropriate for a word embedding evaluation, 3) post-filtering leading to the final set of facets, and 4) facet clustering to facilitate a domain-specific evaluation.

6.3.1 Web Table Filtering and Facet Candidate Generation

Since our goal is to construct an English evaluation dataset, we consider only potential English Web tables. To detect them, we employ a lightweight language identification tool called WhatTheLang[2], which is based on a small fastText model. Moreover, we only incorporate columns with a header occurring at least 10 times. Other columns are unlikely to contain knowledge from the general domain (see Requirement D1). Since numerical values cannot effectively be represented by many word embedding models, we also omit them and only consider text values of Latin letters.

To create facets, we extract all textual header pairs and the associated instance values from the filtered Web tables (see Figure 6.1). All header pairs that occur less than four times are omitted since they are not part of the extended general knowledge, which is expected to be represented in a word embedding model.

[2] https://github.com/indix/whatthelang (Access: 08/03/21)

6.3.2 Check Soft Functional Dependencies

To be useful for a word embedding evaluation, the relations inside a facet should be unambiguous, as stated in Requirement D3. This is the case if the value set V of the facet functional depends on the object set O, e.g., the city depends on the soccer airport's name. However, since Web tables can be noisy, we employ soft functional dependencies (SFD) instead. A similar approach is used in [LB19] to extract knowledge from Web tables. We calculate SFD scores for the pairs of text values taken from the facet generation step described above. Given those pairs in the form of a bag of value tuples $M_{O,V} \subseteq O \times V$ with the multiplicity defined by function $m : \langle O, V \rangle \to \mathbb{N}$, we calculate the SFD for every object $o \in O$ as in Equation (6.1). The value v_{max} refers to the most common value for the object o.

$$SFD(o) = \frac{m(\langle o, v_{max} \rangle)}{\sum\limits_{\langle o,v' \rangle \in M_{O,V}} m(\langle o, v' \rangle)} \tag{6.1}$$

We omit facets with an average SFD smaller than 80%. In addition, all objects with $SFD(o) < 80\%$ are discarded. Moreover, we omit object-value tuples occurring less than 4 times.

6.3.3 Post-Filtering

Not all facet candidates that fulfill the SFD condition are applicable to evaluate word embedding models. Therefore, in a post-processing step, deny lists and pooling are applied.

Deny Lists: Some of the headers are very generic, e.g., "name" and "description". Accordingly, object-value pairs with such header text values model different relation types, making them unsuitable for modeling facets. Moreover, we cannot categorize them. Thus, they are not compatible with Requirement D2.

Pooling: As stated in Requirement D1, we aim to include facts of the extended common knowledge to provide a dataset suitable for a wide range of word embedding models. For this reason, we apply pooling to exclude facets that model relations which are unlikely represented in word embedding models. We select three commonly used large pre-trained word embedding models: Word2Vec (W2V-GN in Table 3.1), GloVe (GV-CC in Table 3.1), and fastText[3]. For each model and each facet, we sample a set of analogy tasks out of terms that can be represented by the model. To solve the analogy tasks, we employ the 3COSADD method (see Section 2.2.5) and restrict the search space to the text values in the value set of the facet V. The size of the task set $|T|$ is chosen depending on the number of possible values $|V|$, where for larger value sets, a larger task set is used. If not more than $\frac{5 \cdot |T|}{|V|-1}$ and less than 50% of the provided tasks are solved correctly by any of the selected word embedding models, the facet candidate is omitted.

[3] fastText Common Crawl (600 billion tokens),
https://fasttext.cc/docs/en/english-vectors.html (Access: 08/03/21)

6.3.4 Categorization

To enable domain-specific evaluation (Requirement D2), we divide the resulting facets into the following broader categories: "sports", "geographic", "music", "movie / video games", "literature", "economy", "technology", and "misc", which are chosen based on a look at the data. The category "misc" is created since some domains only contain a small number of facets.

The categorization is done using a fastText model by providing a list of keywords for each category. For the "misc" category, we provide keywords for all the small categories for which facets exist. The fastText model determines similarity scores of the category keywords to header text values, as well as text values in the object set O and value set V of the facet. Based on those scores, the most similar category is selected. Finally, for each category, a JSON file is compiled containing the respective facets.

6.4 EVALUATION OF POPULAR WORD EMBEDDING MODELS

To demonstrate the value of our approach, we perform in this section an evaluation of popular pre-trained word embedding models. From the results provided by such a process, database users can decide on an embedding model for an SQL query containing word embedding operations (see Section 3.4), and ML developers can select a suitable model to represent the input text. FACETE enables an evaluation of word embedding models on different granularity. While the evaluation presented here focuses on analogy tasks, one can also derive similarity-based metrics or construct cluster-based evaluation tasks from FACETE.

Word Embedding Models We choose four popular English pre-trained word embedding models. This includes the models used in the pooling step in Section 6.3.3: Word2Vec, GloVe, and fastText, as well as a Sentence-BERT model [RG19] called SBert[4], which is fine-tuned on the STS benchmark [CDA+17].

Since Word2Vec and GloVe would suffer from the out-of-vocabulary problem, we tokenize longer text values and average the embedding vectors corresponding to the respective terms in the vocabulary of the embedding model (see Section 2.2.5). For the tokenization process, we use the method described in Section 5.2.1, which takes account of multi-word tokens. SBert and fastText do not have this problem since they inherently include a tokenization process based on word-pieces or n-grams.

6.4.1 Domain-Agnostic Evaluation

Using the word embedding models introduced above, we run an analogy evaluation for the facets in each category of FACETE. The analogy tasks are solved using the 3COSADD method (see Section 2.2.5). However, some facets include a large number of relations from which a quadratic amount of analogy tasks could be constructed. To reduce the number of analogy tasks to the number of relations in the facet, we modify the 3COSADD method: We calculate the centroid of the embedding representations of all text values of

[4] `roberta-large-nli-stsb-mean-tokens`, https://github.com/UKPLab/sentence-transformers (Access: 08/03/21)

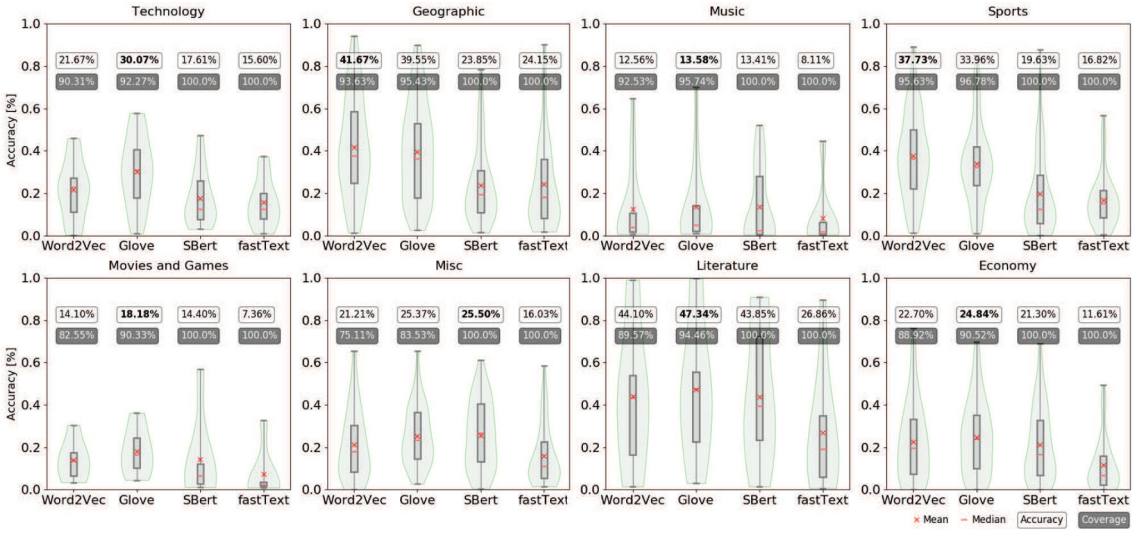


Figure 6.3: Evaluation of Different Domains: Coverage and Distribution of Accuracy Values

the object set O and the centroid of the text values in the value set V. The former is used as the vector a and the latter as vector b in Equation (2.9) in Section 2.2.5. For each facet, we determine the accuracy as the amount of correctly solved analogy tasks and the coverage as the proportion of relations where both text values have a word embedding representation. The distributions of the accuracy values and the average coverage values for each category are shown in Figure 6.3. As one can see, no embedding model outperforms the others in all categories, confirming our hypothesis that word embeddings should be selected on a case-by-case basis. While fastText and SBert have a coverage of 100% due to their processing model, Word2Vec and Glove also achieve high coverages in most of the categories by using the proposed tokenization strategy.

6.4.2 Evaluation of a Single Facet

To compare the representation of a specific property across multiple word embedding models, e.g., for the application in an ML task, one can select a single facet out of our dataset for evaluation. As an example, we use the facet *character→actor* with 111 relations and compare two embeddings. We choose the plain Word2Vec model already used before and a model resulting from applying the RETRO framework of Chapter 5 on the Word2Vec model. For retrofitting, the TMDB movie database (see Section 5.4.1) is used. The retrofitted embedding model achieves an average accuracy of 16.21%, while the plain Word2Vec model achieves 11.71% only. This result suggests using the retrofitted embedding model, even though those low accuracy values indicate that an ML task related to the *character→actor* relation requires more input data.

6.4.3 Evaluation of an Object Set

Instead of evaluating word embeddings on a granularity of facets, one can also calculate the accuracy based on a set of objects relevant to an application. In Figure 6.4, one can see the distribution of the accuracy values from an evaluation of the 100 most frequent city names in FACETE considering six facets in the geographic category with a city object set.

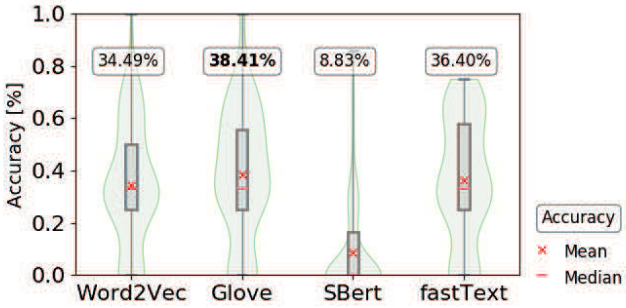


Figure 6.4: Evaluation of City Representations

Here, the accuracy is calculated on the number of values assigned correctly to an object considering the six facets. While Word2Vec achieves the best results in the geographic category (see Figure 6.3), this experiment shows that Glove performs better regarding frequent city terms.

6.5 SUMMARY

To identify a suitable model for an application using word embeddings, a comprehensive evaluation is necessary. In Section 3.7, we already discussed the specific requirements for an evaluation dataset. In this chapter, we reviewed related work regarding those requirements in Section 6.1. We identified that available datasets from the NLP community are designed to evaluate the capabilities of embedding techniques but fail to evaluate to which extend particular domains and word relations are represented in specific pre-trained models. As a solution, we proposed in Section 6.3 a data processing pipeline to automatically generate an evaluation dataset from large table corpora, which we apply to a corpus of Web tables to obtain a comprehensive evaluation dataset called FACETE. The resulting dataset consists of relations organized in multiple facets, which are assigned to eight broader categories (see Section 6.2). In Section 6.4, we demonstrated the application of FACETE in an exemplary evaluation of popular pre-trained word embedding models. The evaluated models perform very differently on the various domains captured by FACETE, and no model achieves the best scores in all domains. These results expose the importance of an application-specific section of the word embedding model.

7

TABULAR TEXT EMBEDDINGS

We presented a comprehensive overview of word embedding applications, where an embedding model is used to represent text in tables in Section 2.4. Many of those applications reuse models pre-trained on text documents. Presumably, several of those applications could profit from a model pre-trained directly on tabular data. In this chapter, we investigate how table embedding models can improve such applications. Therefore, we implement a novel algorithm to train embeddings on a large table corpus and evaluate it on several different tasks involving data in diverse tabular data formats. Results of this comprehensive evaluation and the proposed embedding algorithm presented in this chapter have already been published in [GTGL21]. Our embedding algorithm is designed with respect to the requirements of Section 3.8. Accordingly, the resulting embedding models represent tabular relations between text values in tables. Moreover, the models are schema-aware and applicable for a wide range of applications and tabular data formats.

In the following, we discuss related work on constructing embedding models for tables in Section 7.1. Afterward, Section 7.2 introduces our approach for embedding text values in tables. In Section 7.3, we present several applications of word and table embedding models. Those use the models in unsupervised (see Section 7.3.1) and supervised (see Section 7.3.2) settings. We continue with a comprehensive evaluation of our table embedding techniques in Section 7.4 and conclude in Section 7.5.

7.1 RELATED WORK

In recent years, several research works have investigated techniques for pre-training embedding models on tabular data. However, most of those approaches aim at designing an embedding model for a specific application task or a specific dataset. Using the resulting models on different datasets for different tasks is often not possible or entails some limitations. In the following, we discuss previously proposed table embedding approaches. We categorize the related work into approaches based on the underlying embedding technique. Section 7.1.1 reviews models based on static embedding techniques, and Section 7.1.2 discusses contextualized embedding models. For word embedding models, a comparison between static and contextualized embedding models is discussed in Section 2.2.4

7.1.1 Static Table Embedding Models

Several embedding models have been proposed which use a static word embedding model like word2vec [MSC+13]. Similar to node embedding techniques like DeepWalk and node2vec (see Section 2.2.6), those techniques serialize sequences and apply the word embedding algorithm to them. For serializing sequences from tables, different approaches have been proposed. In [BS17], the text is serialized from a relational database system. The proposed algorithm serializes sequences row by row. In addition, the serializer integrates sequences of rows in other tables referenced by foreign keys. To model relations of text values to the schema, table names and column names are integrated into the sequences. The resulting word embedding model captures relations of text values in the database system. However it is very specific for the database it is trained on. Similarly, in [CPT20], an embedding model is trained on random walks obtained of a graph generated from a specific relational database system. The model does not only generate embeddings for text values in the table body but also for rows and columns, which, however, have no textual bonding. Thus, row and column embeddings for a new table can not be obtained. In [GRE+17], word2vec [MSC+13] is used to generate embeddings for cells in tables later used for blocking. For this purpose, the authors serialize text sequences from table relations. To distinguish cells in the table body and cells in the table header, the proposed serializer adds specific prefixes to the text values. Because of the large diversity of cell values in tables in comparison to terms in natural language texts, the application of this model is limited on the tables it is trained on.

Limitations: Since the techniques described above use traditional word embedding models like word2vec and GloVe, they suffer from the out-of-vocabulary problem. This is especially problematic since text values occurring in tables are often very specific for the table and do not frequently occur in other tables or text documents. Accordingly, it is hardly possible to apply such a pre-trained embedding model on other table corpora as stated in Requirement E1 in Section 3.8. Subword-based embedding models like fastText could be used to overcome this limitation. However, in this case, the embedding technique can not distinguish between subwords occurring in the schema or the body of a table (see Requirement E2).

7.1.2 Contextualized Table Embedding Models

An example of a contextualized Web table embedding model is TabVec [GGS18]. This model trains embeddings for whole tables to solve the table layout classification task (see Section 7.3.2) in an unsupervised way. The model generates random embeddings for words occurring in the table corpus. Then it aggregates the embedding vectors of words in the context of a cell vector. Finally, aggregates of the cell vectors are used as an embedding representation of the whole table. To solve the layout classification task, tables with similar layouts are clustered by an algorithm, which uses the embedding vectors, and the user should manually label the clusters.

Recently, the authors in [GGPS20] propose an embedding model specifically for the classification of cells in spreadsheets. The model uses sentence embeddings [CKS+17] to encode text values in cells, where the tokens of the text values are represented by a pre-trained GloVe model. After encoding the cell values, for each cell, an embedding is derived based on neural networks with an auto-encoder architecture. Instead of calculating an embedding for a cell only on its content, the authors include the content values of the

surrounding cells in the spreadsheet into the calculation of the embedding representation. Moreover, style attributes are incorporated. Since a spreadsheet does not provide information to separate between cells of schema and cells of instance terms, this is not considered by the embedding model.

Building on the success of transformer-based language models like BERT [DCLT19], multiple tabular embedding models [DSL+20, YNYR20, HNM+20, ITMI21, WDJ+21] have been designed using the Transformers architecture [VSP+17]. Specifically, the models TaBERT [YNYR20] and TaPas [HNM+20] conduct a pre-training jointly on text utterances and tables for semantic parsing tasks, e.g., text-to-sql tasks and question answering tasks. For pre-training on relational tables with associated text snippets like page titles and captions, the authors in [DSL+20] propose a model called TURL. A model solely trained on tables called TABBIE is proposed by [ITMI21]. Just recently, the authors of [WDJ+21] present TUTA, an embedding model for tables with a hierarchical header structure. For applying the model, TaBERT, TaPas, TURL, and TUTA serialize tables into sequences. However, TURL and TUTA employ a masked attention mechanism to restrict the attention mechanism of the Transformer model, e.g., TURL utilizes a visibility matrix to focus attention on cells, which are related by the cell structure. TABBIE uses two separate transformer networks to model row-wise and column-wise sequences where the cell embeddings of both transformers are averaged in each layer. For the pre-training of transformer-based table embedding models, dummy tasks are used, which are similar to the masked-language model task (see Section 2.2.3) or the ELECTRA task [CLLM20] in the case of TABBIE pre-training. For applying the ELECTRA task on tabular data, cells are corrupted by replacing the content with a generated content in the input tables, and a binary layer decides if the cell content is part of the original table or not. For the application of the models, the models are usually fine-tuned on specific supervised classification problems. Specifically, applications for those transformer-based models are question answering over tables [YNYR20, HNM+20] and table understanding tasks [ITMI21, DSL+20], e.g., entity linking, column type annotation, and schema augmentation. To incorporate information from language models pre-trained on text, those models either initialize the model parameters with weights of a pre-trained word embedding model [YNYR20, WDJ+21] or encode the input cells with a word embedding model [ITMI21].

To differentiate between different types of text values, [DSL+20] use separate embeddings for text values containing schema-information like cells in the headline and text values in the table body. Moreover, additional input embeddings can be added to the embeddings of the input tokens like segment and positional embeddings in the original BERT model (see Section 2.2.3) to differentiate between text values in the headline and surrounding text like the page title. Similarly, in [HNM+20], the authors use additional embeddings for rows and columns to encode the position of a token in the table. To represent cells, [DSL+20] uses separate entity embeddings added to the input embeddings. In [YNYR20], pooling layers are used to aggregate embeddings of tokens to represent cells.

A different neural network architecture is used in the TCN [WSL+21] model. Here, the authors propose an embedding-based approach with a convolutional neural network architecture for table interpretation tasks. For each value in a table, the network aggregates embeddings of related values in the same table and embeddings of related values in other tables, e.g., syntactical equal text values. The embedding vectors are initialized with embeddings generated with the traditional GloVe embedding model. Afterward, the whole model is pre-trained with a Masked Language Model objective. By taking into account those inter-table connections between values, this model outperforms table models like TURL, which only consider a single table, on various table interpretation tasks.

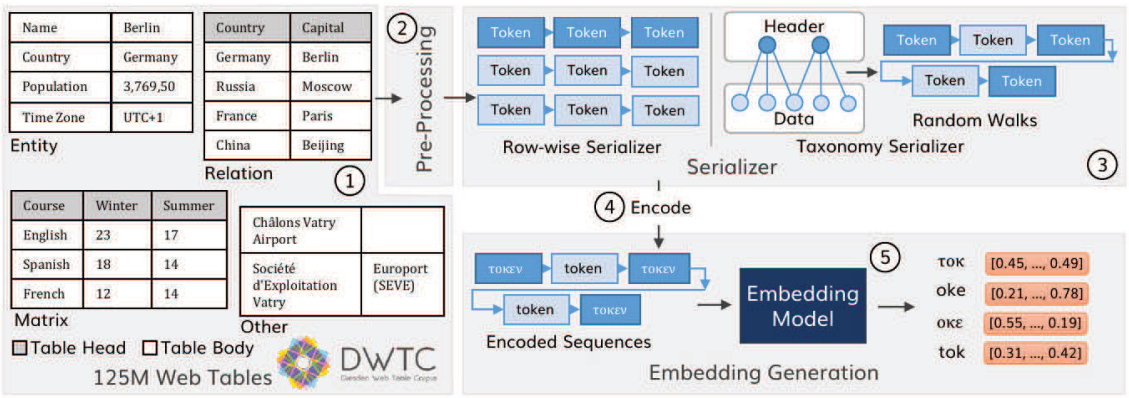


Figure 7.1: Web Table Types and Embedding Process

Limitations: While some of the models, e.g., TURL, are applied in a comprehensive evaluation on different evaluation tasks, the evaluation datasets are very similar to the data used in the pre-training. Frequently, the data used in the tasks constitute a subset of the dataset used for pre-training. Only for TUTA, the pre-trained model is applied to different table types (Web tables and spreadsheets) [WDJ+21], but, also here, both table types are used during the pre-training. It would be interesting to investigate the application of the proposed table model to tasks on other datasets. This is especially relevant for application on datasets that are too small for comprehensive pre-training or contain fewer annotations as the dataset in the pre-training step, e.g., for spreadsheets, it is often hard to identify which cells belong to which table. To apply the approaches above, the models require the whole table as input. When only single text values of cells are provided, the resulting representations are presumably worse. This limits the applicability and thus runs contrary to Requirement E1 in Section 3.8. For instance, cells in spreadsheets often do not belong to a specific table. Moreover, to achieve good performance, Transformer models are usually extended by layers dedicated to a specific classification task, and a fine-tuning is done. Usually, one fully connected layer is added to the Transformer model. However, this procedure might not work well for complex models where the input could not naturally be encoded into a single sequence. Our attempts to integrate a BERT model into a classifier for the layout classification problem (see Section 7.3.2) to encode the rows and columns of tables lead to worse results in comparison to using static embeddings. Another limitation of several Transformer-based models like TaPas and TaBERT constitutes the limited input size, which has to be configured before pre-training. To encode large tables, the authors of [YNYR20] propose to extract content snapshots relevant to an additionally provided utterance, and [LLS+20b] summarizes the input by retaining tokens with high TD-IDF scores. Only TURL produces separate embeddings for schema and instance text values and accordingly satisfies Requirement E2.

7.2 WEB TABLE EMBEDDING MODEL

To fulfill the requirements in Section 3.8, we designed the embedding process visualized in Figure 7.1. For the pre-training, we need a large source of tabular data suitable to train an embedding model. While the tabular data format is pervasive across all fields in academia and industry, we decided to use data from Web tables. Web tables not only constitute a large source of tabular data (hundreds of millions of tables) but also capture a wide range of domains, making them suitable to pre-train an embedding model valuable for a wide range of different tasks. Moreover, a remarkable amount of research has already investigated the problem of mining and cleaning Web

table data [CTT00, WH02, CHZ+08] which led to large already pre-processed corpora of Web tables like the DWTC corpus [EBH+15] containing 125 million Web tables. We utilize the DWTC corpus (1) to derive suitable Web tables. After a pre-processing step (2), we serialize tabular relations in the form of text sequences (3). Those text sequences are encoded by an encoding model (4) with respect to the type (schema text and instance text). Then, a word embedding technique is applied to them (5). While this method simplifies the training of the embedding models, it also provides the possibility to utilize the enormous research made by the NLP community on optimizing embedding models. Moreover, it provides the flexibility of applying different embedding models or even models to be invented in the future.

7.2.1 Preprocessing

In our training, we focus only on English tables. Therefore, we employ the lightweight language classifier WhatTheLang on tables from pages of potential English domains to determine relevant tables. Moreover, we filter out text values with non-Latin letters. For many Web tables, it is easy to separate the schema information from the rest of the data. For instance, header rows are often labeled in the HTML code by a `<thead>` tag. We restrict the corpus to tables where such information is present. Nonetheless, it still encompasses over 40 million tables. Since numerals and special signs do not capture a specific meaning across different contexts they can occur in, we regularize them by replacing them with "@" and "*" characters. This also slightly reduces the large variety of text values observed in Web tables and therefore helps the embedding model to generalize. A similar regularization process was also done in the Web table embedding approach of [GRE+17]. Moreover, spaces are replaced by underscores to coalesce the tokens of a cell into a single token.

7.2.2 Text Serialization

To model different relations according to Requirement E3, we create different serializers to generate sequences from tables:

- **Row-wise Serializer:** The simplest way of serializing tables is to serialize them row by row. Many of the related approaches described in Section 7.1 perform a similar serialization process. A model trained on the resulting sequences potentially represents the relatedness of text values in cells.
- **Taxonomy Serializer:** The taxonomy serializer is designed to produce sequences that capture relations between schema and instance data to build a model which fulfills Requirement E2. For this purpose, we construct a weighted bipartite graph of text values occurring as header and data (non-header) cells. Two text values are connected if one text value serves as a header for the other text value in a table. Thereby, we exclude relations where both text values are equal because this may result from a repetition of the header. Each edge gets assigned the frequency of the relation as weight. Data nodes with only one edge and edges which occur only once (weight < 2) are removed. For header-data node pairs with edges in both directions, the edge with the lower weight is removed. Then, we build upon the idea of DeepWalk [PARS14] to serialize random walks from a graph which are used later to train a language model. We use the graph to construct 100 random walks per node where the transition probability from one node to another is determined by the weight of the edge divided by the sum of all weights.
- **Combined Serializer:** To train a model capturing row-wise and taxonomical relations, the output of both serializers is combined.

Movie Title	Release Date	Language
Godfather	1972	English
Amélie	2001	French
Inception	2010	English

Title	Original Language	Production Year	Genre
Psycho	English	1960	Horror
Good Bye, Lenin!	German	2003	Comedy

Figure 7.2: Unionable Tables

7.2.3 Encoding Model

To represent different types of text values, an encoding model is applied. In our case, we separate between two types: *schema* and *instance*. Previous embedding approaches [TML15, GRE+17] add annotations to words to disambiguate different types. However, this is not applicable for embedding models working with subword information (e.g. ngrams). Therefore, we propose to use a different encoding for each type. Each encoding is defined by an offset. To encode a character, e.g., "a" (unicode 97), the offset, e.g., 255, is added to the numerical unicode representation and interpreted as the resulting sign, e.g., "Š" (unicode 352). This leads to valid encodings since all characters are ASCII signs after preprocessing. In this way, an embedding model can differentiate for every single character if it belongs to a header or instance data text value. Besides our application, this method might also be useful for other typed documents, e.g., documents written in a markup language like TeX and HTML.

7.2.4 Embedding Training

For the training of the embedding model, different word embedding techniques can be applied to the sequences. For the applications in Section 7.3, we use the fastText [BGJM17] approach described in Section 2.2.2 to train embedding models. This leads to flexible embedding models which are able to encode any content of the HTML Web table cell (see Requirement E1). It produces static embeddings, which can be employed for unsupervised and supervised classification tasks.

7.3 APPLICATIONS FOR TABLE EMBEDDINGS

Our goal is to investigate a cross-section of applications for our table embedding models. This encompasses the table union search task (Section 7.3.1), where the model is used in an unsupervised setting, and two supervised tasks where the model is used on Web tables and spreadsheets (Section 7.3.2).

7.3.1 Table Union Search

Tabular data repositories consist of thousands or millions of tables. Data discovery techniques support users in finding useful tables in such repositories. One typical data discovery task is the unionable table search task [MNZ+18]. Thereby, a user wants to find

tables about a specific topic to merge them into a master table. Figure 7.2 displays two examples of incomplete movie tables, which can be used to create a master table for a more complete overview of movies. One can find unionable tables by searching for high unionable column pairs $\langle A, B \rangle$ as described in [NZPM18]. The authors propose three measures for unionability, which can also be combined to an ensemble unionability measure. All three measures define unionability as the probability that the text values in columns A and B are part of the same domain. The best single unionability measure is the natural language unionability based on word vector representations of the column pairs. However, calculating it for all column pairs is not possible for large repositories because of the quadratic growth of the number of pairs. To solve this problem, the authors calculate the mean word vectors for text values in a column as representations of the columns, which can be stored in an index. One can then efficiently search for mean vectors with high cosine similarity to a given column vector which is effective because the cosine similarity correlates with the unionability as shown in [NZPM18]. For implementing the search algorithm, one can use the nearest neighbor search techniques described in Section 4.2. The authors utilize an LSH index specialized for the cosine distance [Cha02]. To generate the word vectors, the authors used a fastText model trained on Wikipedia articles as fT-W in Table 3.1. As shown in Section 7.4.2, the usage of our Web table models can improve the unionability search.

7.3.2 Classification Tasks

We implement ML models for two different classification tasks with tabular input. Those models utilize the pre-trained embedding models to encode the semantic of cells in the tables. First, we introduce a classifier for the layout type of Web tables. Here the input is similar to the tables in the training corpus. Afterward, a classifier for the cell type in spreadsheets is introduced. Here, the tabular input is fundamentally different from the training corpus.

Table Layout Classification

Information stored in tables on the Web has demonstrated its usefulness for several information retrieval tasks, e.g., question answering [SMH+16], building knowledge carousels [CLK+16], data integration tasks like knowledge base completion [CHZ+08], and entity augmentation [ETBL15]. All these tasks require extracting knowledge from Web tables where the table's layout type needs to be determined.

Problem Definition: Web tables can be classified into tables containing genuine content with a semantically significant layout and non-genuine tables, which only align page elements in the HTML document [PHLM01, EBH+15]. To extract content from genuine tables, those tables are further categorized into four layout classes shown on the left of Figure 7.1: ENTITY, RELATION, MATRIX, and OTHER, which is the problem we are focusing on. This is based on the classification done by [LRMB16], which is commonly used in the field. An entity table provides information on a specific entity in the form of an infobox. Relation tables contain multiple entities, where each row lists several attributes of one entity. A matrix table presents property values of the relation between two types of entities mentioned in the the table's first row and first column. While most of the non-genuine tables are already filtered out during the extraction of Web tables, e.g., for the construction of the DWTC corpus [EBH+15], there are still some tables used to layout data that do not fit into one of the categories. Those tables get assigned to category "Other".

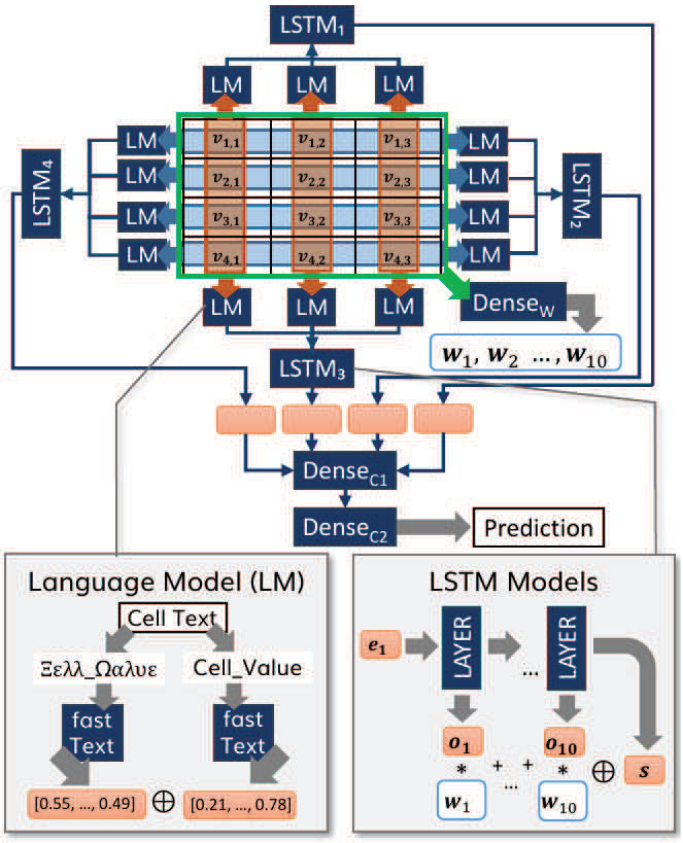


Figure 7.3: Embedding LSTM Model

Model Definition: We developed an LSTM-based model shown in Figure 7.3. First, for every cell content, an embedding is created. If a schema-aware embedding model is used, we create a header cell embedding and a data cell embedding and concatenate them. This is done since it is often not clear for a Web table if a cell contains schema or instance information. For the pre-trained Web table embedding models, we apply the different encoding models for header and data cells to obtain the embeddings. For each direction (up, down, left, and right), a sequence of embeddings is created and fed into an LSTM network. Therefore, the first four embeddings of one row or one column are concatenated to a combined embedding. In addition, each combined embedding is concatenated to itself multiplied element-wise with the mean of all non-zero embeddings of the sequence NZ, which helps the model detecting cell values with an outstanding semantic. The maximal sequence length is set to 10. Small sequences are padded to a length of 10 and a width of 4 with zero vectors. For example, $LSTM_1$ in Figure 7.3 would receive a sequence defined by Equation (7.1), starting with the concatenation of embeddings for $\mathbf{v_{1,1}}, \mathbf{v_{1,2}}, \mathbf{v_{1,3}}$, and a zero vector:

$$\mathbf{e_i} = \mathbf{e'_i} \oplus \left(\mathbf{e'_i} \circ \frac{1}{|NZ|} \sum_{j=1}^{10} \mathbf{e'_j}\right), \quad \mathbf{e'_i} = \mathbf{v_{i,1}} \oplus \ldots \oplus \mathbf{v_{i,4}} \quad (\oplus : \text{Concatenation}) \tag{7.1}$$

$$NZ = \{\mathbf{e'_j} \mid j \in \{1, \ldots, 10\},\ \mathbf{e'_j} \neq \mathbf{0}\} \quad (\circ : \text{Element-Wise (Hadamard) Product})$$

In addition, a concatenation of all embedding sequences is fed into a dense layer $Dense_W$. This layer calculates a probabilistic weight vector $\mathbf{w}$ of size 10. Each of the four LSTM networks calculates a set of 10 outputs which are aggregated by using the weights w and concatenated with the final status value of the LSTM. This enables the classifier to weight the importance of the cell positions with respect to the table content. The results

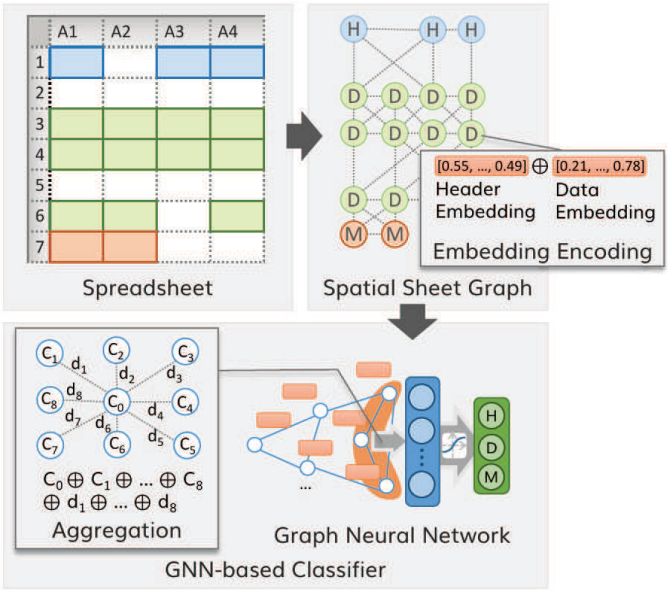


Figure 7.4: Cell Classification Model

of those calculations are fed into two dense layers $Dense_{C1}$ and $Dense_{C2}$ to perform the final classification. To predict one of the four classes, we use a one-vs-all strategy [RK04] by training the network four times for each class. We then determine the prediction by taking the maximum value, which seems to outperform a multi-class prediction with softmax activation.

Spreadsheet Cell Classification

Spreadsheets are widely used in industry to organize data. While this data format allows humans to easily handle and modify data, it is hardly machine-readable. To extract data from spreadsheets, it is necessary to determine whether cells contain instance data, schema information (header cells), or metadata [KTRML16]. While this classification itself helps to extract data from spreadsheets, more complex spreadsheet annotation tasks like table recognition can be done based on this classification [KTRL19].

Problem Definition: The smallest structural unit of a spreadsheet is a cell. Usually, the creator of a spreadsheet organizes data in a sheet in the form of tables, where one spreadsheet can contain multiple tables. Those cells can be classified based on their role in the layout of their tables. While different classification schemes can be found in the literature, we distinguish three main types according to the classification of [KTRL19]: HEADER (H), DATA (D), and METADATA (M). Headers are captions of table columns describing the cells below them. In practice, tables in spreadsheets often contain multi-row headers. Metadata cells provide additional information about a table or the spreadsheet as a whole. Footnotes and comments are typical examples of the content of metadata cells. In contrast to the other types, Data cells do not describe other cells but contain the actual instance data. We consider the cell classification task as a supervised classification problem. Thereby, an algorithm is supplied with a set of spreadsheets where the label of each cell is provided. Based on the provided labels, it should derive a classification function to label cells in unseen spreadsheets.

Model Definition: To solve the cell classification problem, we propose a GNN-based classification model. Figure 7.4 depicts its architecture. The classifier can be trained on a corpus of multiple spreadsheets. Similar to [KTRL19], we construct a graph $G = (V, E)$ for each sheet to represent its spatial structure. The nodes V of this undirected graph represent the cells. Neighboring nodes up to a distance of three cells are connected by edges E annotated with the direction p and distance δ. Afterward, a graph neural network (GNN) is trained on the graphs. This GNN uses a pre-trained embedding model to obtain an input feature vector of the cell content for each node. It is defined by a list of hidden layers $H_i^{m \times m'}$ and a message-passing function $\mu : E^8 \rightarrow \mathbb{R}^{m'}$. The μ function takes as input all edges connected to a node C_0 and generates for this node an input vector for the next layer:

$$\mu(\langle C_1,C_0\rangle, \ldots, \langle C_8, C_0\rangle) = \\ M(C_0) \oplus \ldots \oplus M(C_8) \oplus \upsilon(d\langle C_1, C_0\rangle) \oplus \ldots \oplus \upsilon(d\langle C_8, C_0\rangle) \tag{7.2}$$

First, the function obtains the embedding vectors of node C_0 and the neighboring nodes $C_1, \ldots, C_8$ in the eight directions (see Figure 7.4) denoted by $M(C_0), \ldots, M(C_8)$. To construct the initial embeddings, we use a pre-trained embedding model. For a schema-aware model, a header embedding and an embedding representing an instance text value are obtained and concatenated (see Figure 7.4) for each cell. Then, the embedding is concatenated with the mean embedding vector of the row and the mean embedding vector of the column. For the intermediate layers, $M(C_0), \ldots, M(C_8)$ constitute the output vectors of the last layer. The function μ concatenates the received embedding vectors with the encoding of the distance values. Therefore, the encoding function υ calculates the inverted unary encoding of the distances $\delta - 1$. For example, for a direct neighbor ($\delta = 1$), the value 0 is encoded by the unary encoding 000, which leads to the representation 111 after inverting. In the same way, a distance $\delta = 3$ is encoded by 001, the inverted unary encoding of 2 (unary: 110). For missing neighbors, zero embedding vectors and the distance encoding 000 are used. To calculate the output X^{i+1} of each layer H_i, μ calculates the input vectors from the features of the previous layer stored in the matrix N^i, the matrix H_i is multiplied, and the sigmoid activation function σ is applied to the result. Thereby, $\mathcal{N} : V \rightarrow E^8$ denotes the neighbor function that returns the incoming edges of the given node.

$$X^{i+1} = \sigma(N^i \cdot H_i) \qquad N^i = \begin{bmatrix} \mu(\mathcal{N}(X_1^i) \\ \vdots \\ \mu(\mathcal{N}(X_n^i) \end{bmatrix} \tag{7.3}$$

The proposed GNN can be seen as a specific variant of GraphSage [HYL17]. However, in our case, the message-passing function receives a fix-sized list instead of an unordered set of vectors.

7.4 EVALUATION

In this section, we present the results of our evaluation of table embedding models. Therefore, we train four different embedding models with our proposed embedding process. In addition, we use various pre-trained word embedding and table embedding models as baselines. In the following, we give an overview of the pre-trained models. To evaluate if models are schema-aware, we conduct an intrinsic evaluation in Section 7.4.1. Besides, we evaluate the performance of our embedding models in the application tasks described in Section 7.3. Specifically, Section 7.4.2 discusses the evaluation results for table union search, Section 7.4.3 presents the evaluation of the proposed table layout classification model, and the evaluation results of the spreadsheet classification are shown in Section 7.4.4.

Table Embedding Models The embedding process of Section 7.2 allows generating different embedding models by using different configurations. We trained the following four table embedding models:

- W_{plain}: We obtain a basic Web table model by using the row-wise serializer without a pre-processing step. Since types are not annotated without pre-processing, the serialize does not separate between schema and instance terms.
- W_{row}: For this model, we use the row-wise serializer with pre-processing. Here, the serializer separates between schema and instance terms.
- W_{tax}: This model is trained on the output of the taxonomy serializer after pre-processing and encoding the sequences according to the cell type.
- W_{combo}: For this model, we use the combined serializer after pre-processing and encoding.

The embedding technique, which we choose for training the models, is fastText [BGJM17]. For supervised tasks, we train the models to produce vectors with only 64 dimensions to increase the efficiency of the ML models and prevent overfitting. For the unsupervised tasks of Section 7.4.1 and Section 7.4.2, the models are configured to produce 150-dimensional vectors.

Baseline Embedding Models We compare our Web table models with several baseline embedding models. Those encompass three popular pre-trained word embedding models mentioned in Table 3.1 in Section 3.2, a pre-trained contextualize BERT model, and two contextualized table embedding models. For some experiments we only use the fastText word embedding model (fT-W) since fastText is also used for training the Web table embedding models. Thus, it can best be compared to our models. Moreover, fT-W has also been used by [NZPM18] for the table union search (see Section 7.3.1). In the following we describe, how we used the models to encode text values for the investigated application:

- **fastText Wikipedia (fT-W):** While this model is trained on different input sequences, it behaves similarly to W_{plain}. Thus, it is able to directly encode a text value in a cell to a static vector representation.
- **Word2vec Google News (W2V-GN):** This model is trained with the Skip-Gram model [MSC+13, MCCD13] (see Section 2.2.2). Thus, it has a fixed vocabulary, and many text values can not directly be modeled. To model text values consisting of multiple tokens, we use the tokenization strategy described in Section 5.2.1 and average the vectors corresponding to the tokens.
- **GloVe Common Crawl (GV-CC):** This model trained with GloVe [PSM14] (see Section 2.2.2) also suffers the out-off-vocabulary problem. Here, we also apply the procedure of Section 5.2.1 to model text values.
- **BERT-Large, Uncased, Whole Word Masking (BERT-Large):** This model[1] is trained with the contextualized embedding technique BERT [DCLT19] described in Section 2.2.3. Usually, BERT models are designed for supervised tasks where the model is fine-tuned. However, fine-tuning is not possible in unsupervised settings. Moreover, in deep learning models with a complex architecture, e.g., the models described in Section 7.3.2, it might not be effective to integrate the whole BERT model.

[1] https://github.com/google-research/bert (Access: 08/17/21)

Therefore, we use the bert-as-a-service framework[2] with the standard configuration to obtain static representations from a BERT model for unsupervised tasks. In this way, a text value is tokenized, the pre-trained model is applied, and the embeddings obtained from the last but not least hidden layer are averaged to obtain a static vector representation.

- **TaPas-Base (TaPas):** This model[3] is pre-trained with the contextualized table embedding model TaPas [HNM+20] described in Section 7.1.2. Since this model is proposed for semantic parsing tasks over tables, e.g., question answering, it expects a text utterance and a table as input. We use this model for the table union search task (see Section 7.3.1) for encoding table columns. For our applications, the input is usually different. Thus, we provide only an empty string as utterance to the model and the input column. For inputs without a column header, an empty header string is provided. Since TaPas can only encode a limited number of tokens, we truncate all text values longer than 100 characters and provide only the first 100 rows to the model. After applying the model, we average the embeddings of the last hidden layer to obtain a static representation.
- **TURL:** The authors of the table embedding approach TURL [DSL+20] also provide a pre-trained table model[4]. However, here no tokenization function is implemented for pre-processing a tabular input for the model. Thus, we implemented a function[5], which encodes the input in a way similar to the data loaders implemented for the evaluation tasks in the paper. However, it could be further investigated if other tokenization strategies lead to superior results. For the table union search task, the column provided is encoded as a core column. If a header is provided to the model, we encode this as a header line with the TURL model. To obtain embeddings for the header the corresponding embeddings of the last hidden layer are averaged. To obtain an embedding for data cells, we take the entity embedding of the last hidden layer.

7.4.1 Intrinsic Evaluation

In the following, we evaluate the embedding models regarding their representation of relations between schema and instance text values. Therefore, we conduct an intrinsic evaluation in the form of an unsupervised link prediction.

Dataset: To obtain the text value relations, we build a taxonomy graph of INSTANCE_OF and SUBCLASS_OF relations. Therefore, we utilize a reduced version of the YAGO 4 ontology [TWS20] consisting of entities present in the English Wikipedia[6] and labels from schema.org[7] ontology [GBM16]. We use the graph to obtain for each instance all related classes. However, we ignore the generic relations to the root class "Thing", which is valid for all instances and not expected to be represented by the embedding models. Then, we randomly sample $10,000$ valid instance-class relations and $10,000$ relations not present in the dataset (negative sample set) together with their respective labels. An embedding model $M : s \rightarrow \mathbb{R}^d$ should assign similar vectors to the labels of entries in the valid relation set E_v and dissimilar vectors to labels in pairs of the negative sample set E_n. Ideally,

[2] https://github.com/hanxiao/bert-as-service (Access: 08/17/21)
[3] https://huggingface.co/google/tapas-base (Access: 08/17/21)
[4] https://github.com/sunlab-osu/TURL (Access: 08/17/21)
[5] https://github.com/guenthermi/table-embeddings/blob/transformer-table-models/unionability_search/turl_embedding_model.py (Access: 08/19/21)
[6] https://yago-knowledge.org/data/yago4/en/ (Access: 02/05/2021)
[7] https://schema.org/docs/developers.html (Access: 02/05/2021)

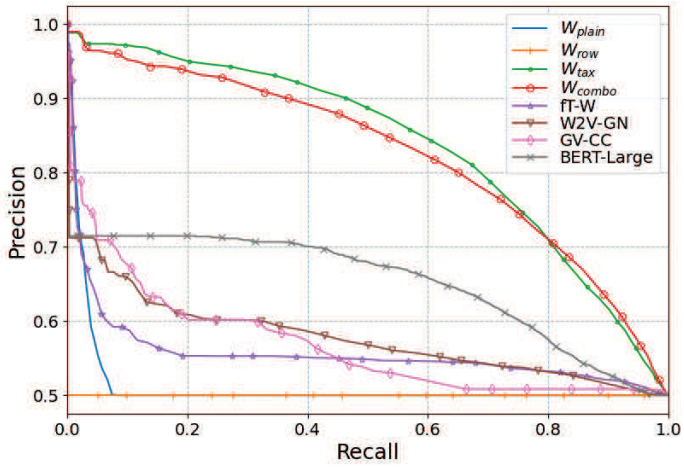


Figure 7.5: Precision-Recall Curves of Instance-Of Relations

a threshold th separates the cosine similarity values of all valid pairs from the values of all negative sample pairs:

$$\forall \langle v_1, v_2 \rangle \in E_v : sim_{cos}(M(v_1), M(v_2)) > th \qquad (7.4)$$
$$\forall \langle n_1, n_2 \rangle \in E_n : sim_{cos}(M(n_1), M(n_2)) \leq th$$

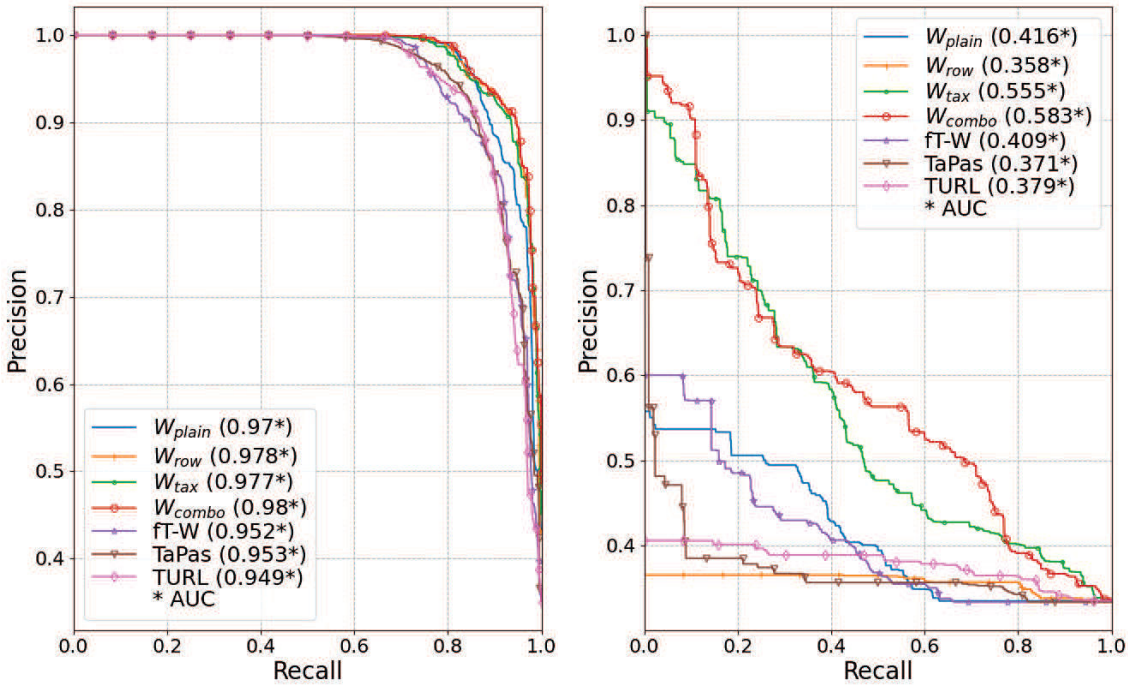
Higher thresholds lead to more undetected relations (false negatives) and lower values to more falsely detected relations (false positives). To evaluate an embedding model, we calculate all cosine similarity values and sort the pairs accordingly. For each position in the resulting list, one can calculate precision and recall. For the Web table embedding models, class terms are encoded with header embeddings and instance terms with data embeddings.

Results: Figure 7.5 presents the resulting precision-recall curves for the Web table embedding models, the fastText word embedding model, and the other popular word embedding models. The area under the curve (AUC) for the schema-aware Web table embedding models is much larger than the AUC of the word embedding models, W_{plain}, and W_{row} model.

7.4.2 Table Union Search Evaluation

To evaluate the unionability search (see Section 7.3.1), we use a benchmark[8] introduced by [NZPM18]. It contains annotated unionable columns of tables created from real-world data of an open data repository. To evaluate embedding models, we sample $10,000$ unionable column pairs and $20,000$ non-unionable column pairs from this dataset. Afterward, we use the embedding model to obtain a vector for each text value in the columns and calculate for each column the mean embedding vector. Then, we calculate the cosine similarity values for each pair in the sample sets to create a ranking of the pairs where high values should indicate unionability. Figure 7.6a shows the interpolated precision-recall curves for the rankings obtained by different embedding models. The best ranking is obtained by the Web table embedding models with pre-processing followed by the baseline Web table embedding model W_{plain}. A fastText model as used by [NZPM18]

[8] https://github.com/RJMillerLab/table-union-search-benchmark (Small) (Access: 02/05/2021)



(a) Column to Column

(b) Column to Column Header

Figure 7.6: Precision-Recall Curves of Unionability Search

constructs a less optimal ranking. The transformer-based table embedding models TaPas and TURL only obtain similar results like the fastText model. This might be the case since those models are not designed for such unsupervised tasks. A fine-tuning with the unionability benchmark might be beneficial to obtain better static embedding vectors.

For some applications, only the schema of the candidate tables is available since retrieving the data of all tables is not possible or too costly. To cope with this setting, we obtain embeddings of the headers of the candidate columns instead of mean embeddings to pre-filter candidates. For W_{tax}, W_{row}, and W_{combo}, we use the encoding model for header cells. Figure 7.6a shows the result obtained in this setting. Here, also W_{tax} and W_{combo} achieve the best performance on this much harder retrieval task. W_{plain} and $fT - W$, which can not implement a different encoding for header cells, achieve lower precision values. TaPas and TURL achieve comparably worse results. Since those models are designed to encode a whole table, a single header cell might not be a suitable input. W_{row} achieves the lowest AUC score since it does not model relations between header and data cells.

7.4.3 Table Layout Classification

For our evaluation of the layout classification model (see Section 7.3.2), we use a human-labeled dataset similar to the one used by [EBH+15]. The novel dataset contains more tables than the previously used table collection.

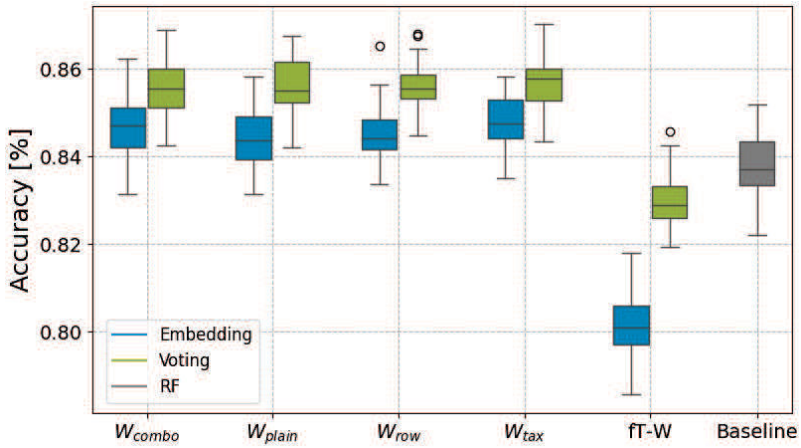


Figure 7.7: Layout Classification Accuracy

	ENTITY			MATRIX			RELATION			OTHER		
	Pre.	Rec.	F1	Pre.	Rec.	F1	Pre.	Rec.	F1	Pre.	Rec.	F1
RF	**87.93**	87.44	87.67	**84.66**	75.16	79.58	80.16	89.79	84.70	**85.19**	68.37	75.80
W_{plain}	87.02	90.07	88.51	77.14	83.49	80.15	86.05	84.91	85.46	81.37	72.79	76.78
W_{row}	87.74	90.03	**88.85**	76.86	**84.25**	**80.30**	**86.27**	84.48	85.34	80.65	**73.92**	77.07
W_{tax}	86.97	**90.47**	88.66	78.78	81.78	80.15	85.46	86.35	**85.88**	84.36	72.66	**78.03**
W_{combo}	87.64	89.74	88.66	78.53	81.34	79.81	85.15	**86.61**	85.85	83.32	73.06	77.80
fT-W	80.75	88.18	84.26	79.94	69.93	74.55	80.08	84.07	82.00	79.54	63.40	70.45
Voting Classifier												
W_{plain}	88.08	**90.79**	89.40	80.76	82.31	81.49	**85.65**	87.76	86.69	84.54	**72.83**	78.21
W_{row}	88.38	90.76	**89.54**	81.10	**82.36**	**81.65**	85.56	87.64	86.57	83.99	73.10	78.11
W_{tax}	87.94	90.96	89.41	81.34	80.92	81.04	85.56	87.64	86.57	83.99	73.10	78.11
W_{combo}	**88.45**	90.30	89.35	80.90	80.29	80.52	84.76	**88.67**	86.65	85.59	72.61	78.53
fT-W	83.74	90.53	86.97	**84.02**	72.09	77.54	81.53	87.43	84.36	85.39	66.41	74.64

Table 7.1: Table Layout Classification Results

Human-Labeled Dataset: The dataset is created by experts using a Web tool specifically designed for this purpose[9]. We published the dataset[10] to enable others to reproduce our result and further investigate the table layout classification problem. It contains 5,777 tables. We consider only the English tables since our embedding model is trained on English data. Those comprise 75% of the dataset.

Baselines: We compare the results of our Web table embedding models to the random forest model proposed in [EBH+15], which serves as a baseline (RF). To train and evaluate the random forest model, we extract the same features engineered by the authors for all tables in the dataset. Besides, we use the pre-trained Wikipedia fastText model (fT-W) introduced above together with our LSTM-based classifier to compare it to our Web table embedding models.

Model Training: All models are trained 50 times on 1,732 samples (40%). The accuracy is validated on 2,165 random samples (50%). 10% of the data are ignored and only

[9] https://github.com/jgonsior/dwtc-table-manual-classificator (Access: 02/05/2021)
[10] https://wwwdb.inf.tu-dresden.de/misc/web-table-embeddings/labeled_layouts/data.db.gz (Access: 03/16/2021)

used for monitoring the training process. We train the LSTM model with the novel Web table embeddings and the fT-W model. As described in Section 7.3.2, we use the embedding models to encode text values in the cells of the tables. However, for the English fastText model , we do not encode the text values with a special alphabet and obtain only one embedding per text value. The random forest classifier is trained with the hand-crafted features of [EBH+15]. In addition, we investigated combinations of the hand-crafted features and embedding features. However, concatenating the manually designed structured features with the embedding features to train a classifier leads to poor performance. This might be the case because random forest classifiers generally perform better on structured data, and neural network classifiers achieve better results on embedding features [AP21]. Thus, we combine the classifiers themselves by a simple voting method that averages the prediction of both classifiers.

Results: Figure 7.7 presents the distributions of the accuracy values obtained in the evaluation of the models. The LSTM model outperforms the baseline when using Web table embeddings, whereas it delivers the lowest accuracy with word embeddings. In combination with the RF baseline (Voting), the accuracy values of all embedding models improve. A closer look reveals that the schema-aware models W_{tax} and W_{combo} perform slightly better than the other Web table embedding models. The mean accuracy values of W_{tax} ($p = 0.005$) and W_{combo} ($p = 0.049$) are significantly higher than the mean accuracy of W_{row}. Table 7.1 presents the results for each layout type. The Web table embedding models achieve better F1 scores than the RF classifier and the fT-W model on all classes. W_{tax} and W_{combo} perform slightly better on tables with classes *Relation* and *Other* compared to W_{row} and W_{plain}.

7.4.4 Spreadsheet Cell Classification

In the following, we present the evaluation of our GNN-based model for spreadsheet cell classification (see Section 7.3.2) with different embedding models. We conduct this evaluation on a dataset of annotated spreadsheets. A random forest classifier working with manually selected structured features [KTRML16] serves as a competitive baseline.

Dataset: The DECO dataset published in [KTR+19] provides a comprehensive collection of 854 sheets with annotations for all non-empty cells in the sheets. Cells are annotated with seven labels. According to [Koc20], those are assigned to the three meta labels mentioned in Section 7.3.2: DATA (DATA, DERIVED, and GROUPHEAD), HEADER (HEADER), and METADATA (METATITLE, NOTES, and OTHER). While the dataset contains 1.7M labeled cells, most of the cells ($> 94\%$) belong to the "Data" metaclass, and only very few labels are Headers ($< 1.7\%$). Therefore, we evaluate our classifier only on 181 small sheets with up to 100 cells, which have a more even distribution of the labels (Data: 78%, Header: 13%, Metadata: 9%).

Baseline: For the cell classification problem, [KTRML16] proposes a comprehensive set of hand-crafted features. Further, the authors in [KTRML16] evaluate several common classifiers on those features. As the random forest classifier achieves the highest accuracy, we use it together with the features extracted for the DECO dataset features[11] as a baseline approach in our experiments. In addition, we also used the pre-trained Wikipedia fastText model (fT-W) as a baseline embedding model for training our proposed GNN-based model.

[11] https://drive.google.com/file/d/1_xOEBfryuFzOUU6bHRZOJl8m1bYko4Wg/view?usp=sharing (Access: 02/05/2021)

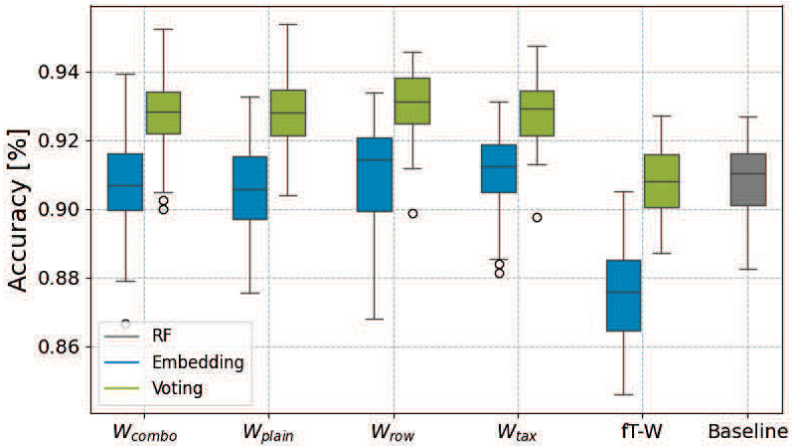


Figure 7.8: Cell Classification Accuracy

	DATA			HEADER			METADATA		
	Pre.	Rec.	F1	Pre.	Rec.	F1	Pre.	Rec.	F1
RF	92.4	**97.6**	94.9	86.3	81.4	83.6	**78.1**	46.1	**57.4**
W_{plain}	**94.8**	95.2	94.9	89.4	87.7	88.5	56.5	**54.9**	54.8
W_{row}	94.3	96.0	95.2	**89.8**	88.0	**88.8**	60.0	50.1	53.2
W_{tax}	94.1	96.5	**95.3**	88.3	87.9	88.0	64.3	48.2	53.9
W_{combo}	94.2	95.7	95.0	88.5	**88.1**	88.3	60.0	50.3	53.2
fT-W	92.2	93.6	92.9	81.9	79.9	80.8	53.0	46.2	48.1
Voting Classifier (GNN-Model + Random Forest)									
W_{plain}	**94.5**	97.9	96.2	92.2	88.8	90.4	73.2	**53.6**	**61.2**
W_{row}	94.3	**98.5**	**96.3**	**92.7**	89.0	**90.8**	77.8	49.9	60.0
W_{tax}	94.0	**98.5**	96.2	91.5	88.6	90.0	**79.3**	48.6	59.6
W_{combo}	94.1	98.3	96.1	91.8	**89.1**	90.4	77.4	49.9	59.7
fT-W	92.6	97.3	94.9	88.7	82.9	85.6	71.5	46.4	55.3

Table 7.2: Evaluation Results for Layout Types

Model Training: For the training of each model, we sample 50 times 72 sheets as a training set and 90 sheets for testing. In this way, the models are required to classify cells from unseen sheets on the test set. For the random forest classifier, we create feature vectors from the manually designed features. Therefore, we encode categorical features by one-hot encodings. To apply the GNN model, we create graphs for all sheets in the training and testing set. The graphs are annotated with feature vectors obtained by an embedding model for the content of each cell. Therefore, we use our three Web table embedding models and the fastText model fT-W. As described in Section 7.3.2, we obtain for schema-aware models two embeddings for each type and concatenate them. For W_{plain} and fT-W, only one embedding is obtained for each cell. The training of the GNN is done in 150 epochs with the Adam optimizer [KB15]. In addition, we again use a voting method to combine the random forest classifier with the GNN-based classifiers.

Results: Figure 7.8 presents the distributions of the accuracy values of the classification models. As one can see, GNN models trained on Web table embedding models outperform the GNN model trained with word embeddings and achieve similar results as the random forest classifier. All GNN models can profit from a combination with the random forest classifier via the voting method. Table 7.2 shows the class-specific mean values for

precision, recall, and F1 score. The random forest model is better at recognizing metadata cells. This might be the case since metadata cells can be identified by the style and font features that the GNN model does not consider. In contrast, the GNN models obtain much better F1 scores for header cells. The voting classifiers achieve the best F1 scores for all classes.

7.5 SUMMARY

In this chapter, we discussed the training of embedding models on tabular data. Therefore, we reviewed techniques proposed for embedding tabular content in Section 7.1. Those include embedding models, which produce static embedding representations for cells in tables, and contextualized table embedding models, which are in the primary focus of current research in this field. Then, we presented our techniques for serializing tables for training embedding models in Section 7.2. Thereby, we take into account our requirements defined for a table embedding technique in Section 3.8, which are not completely covered by the related work. Furthermore, we implement novel solutions for data discovery tasks like table union search, table layout classification, and spreadsheet cell type classification (see Section 7.3). We used the proposed embedding procedure to pre-train several table embedding models. We then conducted in Section 7.4 a comprehensive evaluation on them. The results show that those models capture semantic relations between schema text and text in instance data not recognized by word embedding model trained on text (Section 7.4.1). In Section 7.4.2, we compared our embedding models to word embedding models and transformer-based table embedding models in the table union search tasks. In contrast to those recently proposed table embedding models, our models are designed to obtain static embedding representations and do not aim at encoding a whole input table. In the proposed setting, our static table embedding models outperform those complex models. This indicates that those models are less useful compared to static embedding techniques in unsupervised settings. An interesting research direction would be to investigate how fine-tuning methods could be employed to produce better static embedding representations of cells with those models. We further evaluated our embedding models together with our supervised ML models for table layout classification (see Section 7.4.3) and spreadsheet cell type classification (see Section 7.4.4). The results demonstrate that our models are useful for a wide range of applications and a diverse set of tabular data formats. Our findings suggest that models pre-trained on tables constitute competitive alternatives to pre-trained embeddings on text. This is especially the case for applications where tabular and or schema information has to be modeled.

8 CONCLUSION

In this last chapter, we summarize the content of this thesis to provide a general overview of the motivation, challenges, and results of our research in Section 8.1. Further, Section 8.2 discusses directions for future work.

8.1 SUMMARY

The focus of this thesis is the integration of techniques based on word embedding models into relational database systems. Accordingly, we began with a comprehensive overview of word embedding algorithms and their application for data management tasks in Chapter 2. In this context, we also discussed the differences between static and contextualized word embedding techniques. The focus in this thesis lay on static embedding representations and their integration in database systems. However, multiple methods have been proposed to generate static representations with contextualized word embedding models. We described the properties of word embedding representations and techniques for the evaluation of word embedding models. We also introduce techniques like node embedding methods and graph neural networks, which are closely related to word embedding techniques and have also inspired our research. Finally, we gave an overview of applications of word embedding models on tabular data, which serve as inspiration for our research and can potentially benefit from it.

Chapter 3 gave a detailed overview of our goals for integrating word embeddings into database systems. Those have a two-fold character. On the one hand, we intended to use word embeddings to extend the text analysis capabilities of database systems. On the other hand, our research focus encompasses the development of novel algorithms to improve applications of word embedding models working with tabular data. We surveyed existing systems for managing word embedding models and analyzed popular word embedding models to identify requirements for handling word embeddings in database systems. To further refine our research focus and challenges, we designed an system overview with five components. Those components encompass the (a) novel word embedding operations, (b) fast access methods for embedding representations, (c) novel optimizing algorithms for embedding representations of text values in the database, (d) solutions for recommending pre-trained embedding models, and (e) embedding techniques for tabular data. For all of them, we gave motivation and analyzed the challenges that lead to concrete requirements.

The subject of Chapter 4 is the extension of the capabilities of database systems with word embedding techniques. In this context, we described the system architecture and implementation of our PostgreSQL extension FREDDY, which integrates novel word embedding operations in the DBMS, and the associated Web interface. Moreover, we investigated techniques for optimizing the run-time performance of the embedding operations. Since most word embedding operations can be executed via the kNN-Join operations, we review the scientific literature on kNN search. After discussing the advantages and disadvantages of current approaches for kNN search in database systems applicable for our problem, we introduced our novel flexible algorithm for approximate kNN-Joins, which we integrated into the FREDDY extension. This algorithm builds on the related work but implements novel optimization techniques to better fit the requirements of word embedding operations. Moreover, we integrated a novel kNN-Join algorithm for binary word embedding representations. In our evaluation, we demonstrated that our proposed optimizations effectively increase the performance of the kNN-Join algorithm and the efficiency of the kNN-Join operation for bit-vectors.

In Chapter 5, we investigated techniques for optimizing embedding representations of text values in relational database systems. Therefore, we surveyed embedding techniques to create vector representations that capture semantic properties of text values obtained from relational data and text documents. Based on the concept of retrofitting, we designed a novel algorithm with the specific goal of optimizing embedding representations of text values in database systems. This algorithm incorporates relational information obtained from the alignment of text values in the database and foreign key relations into embedding representations derived for the text values from pre-trained word embedding models. In addition, we designed an online algorithm to support instant optimization of embedding representations of text values added to the database with insert queries. For our evaluation, we developed simple feed-forward and complex graph neural network classification models for various classification tasks. Our results demonstrate how those ML models profit from the optimization of embedding representations.

For the recommendation of pre-trained word embedding models, Chapter 6 investigates techniques for domain-specific word embedding evaluation. Our review on word embedding evaluation techniques indicates that an evaluation based on currently available datasets is not appropriate for deciding on a pre-trained word embedding model for a particular domain. To solve this problem, we developed an extraction framework to construct comprehensive word embedding evaluation datasets from a large collection of tabular data. We employ our approach on a large corpus of millions of Web tables to obtain a dataset covering a wide range of domains and relation types. Because of the the dataset's facet-structured architecture, one can obtain a detailed report on the applicability of word embedding models for specific domains and applications. Our evaluation of common pre-trained word embedding models on this dataset indicates that there is not a single best model for all applications.

Finally, Chapter 7 is concerned with methods for training embedding models on tabular data. Here, we investigated static and contextualized embedding techniques for tables and their limitations. Current techniques cannot differentiate between cells modeling schema and instance data, suffer from the out-of-vocabulary problem, or are not effective for obtaining static embedding representations. We presented a novel embedding approach covering all those aspects. In our evaluation, we demonstrated that our approach effectively models relations between schema information and instance data. Moreover, we showed that models pre-trained with our approach are applicable for several different application tasks and tabular data formats.

8.2 DIRECTIONS FOR FUTURE WORK

Developing embedding models for text, tabular data, and other data formats is still an active research field. Moreover, embedding techniques and their applications for database systems and tabular data gain more and more attention in the data management community. Accordingly, further integrating word embedding techniques and database systems is definitely an interesting topic for future research. In the following, we present some promising directions, which arise from our work on this topic:

Novel word embedding operations: The novel word embedding operations, which we integrate into FREDDY, already offer a wide range of novel text analysis capabilities. However, word embedding models offer many other query opportunities one could integrate into relational database systems. Specifically, systems like TabVec [GGS18] allow users to label objects based on the clustering of embedding vectors. More comprehensive clustering operations could support such applications, e.g., for fitting a clustering to a given set of labels. Moreover, techniques like [MLS13] allow the projection of embedding representations between different models. Integrating such techniques into database systems would enable novel operations like cross-lingual similarity search. Besides, contextualized embedding models offer a wide range of new query possibilities, e.g., high performance in tasks like question answering [DCLT19, HNM+20] and table summarization [GCS19]. First attempts to integrate those capabilities into database systems have been made by NeuralDB [TYS+21], which offers an efficient query engine for answering natural language queries over a database storing a large document collection.

Deep integration of kNN-Join algorithm: Our novel kNN-Join algorithm already achieves large performance gains in comparison to a naïve algorithm. However, deeper integration of our proposed index structures into PostgreSQL could lead to further performance improvements. In [YLFW20], the authors propose a deep integration of index structures based on product quantization into PostgreSQL, which leads to significant performance gains. By further integrating our index structures into PostgreSQL, one could also address the optimization of queries involving multiple kNN-Join operations and embedding models. Moreover, in our search algorithm, one could integrate techniques for an automatic configuration of the hyperparameters for specific precision and execution time requirements. In this way, the usability of the algorithm could be increased.

Support a wider range of embedding models: While this thesis restricts its focus on embedding models for text values in the database, it would be useful to support embedding representations for other data types. While some techniques developed in this thesis, like the kNN-Join algorithm, could be easily used for vector representations like image descriptors [Low04] and node embeddings (see Section 2.2.6), different embedding representations might require different operations. Moreover, our relational retrofitting algorithm can not take non-textual values into account. We presume that embedding representations of text values could be improved by techniques for including numerical values and embedding representations of non-textual content into RETRO.

BIBLIOGRAPHY

[ABF20] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. *Information Systems*, 87:101374, 2020.

[ACD02] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A System for Keyword-Based Search Over Relational Databases. In *Proceedings 18th International Conference on Data Engineering*, pages 5–16. IEEE, 2002.

[AG16] Oded Avraham and Yoav Goldberg. Improving Reliability of Word Similarity Evaluation by Redesigning Annotation Task and Performance Measure. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 106–110, 2016.

[AKLS15] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Cache Locality is Not Enough: High-performance Nearest Neighbor Search with Product Quantization Fast Scan. *Proceedings of the VLDB Endowment*, 9(4):288–299, 2015.

[ALM17] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A Simple but Tough-to-Beat Baseline for Sentence Embeddings. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[AM93] Sunil Arya and David M Mount. Approximate Nearest Neighbor Queries in Fixed Dimensions. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 271–280. Citeseer, 1993.

[AMCG15] Abdulaziz Alghunaim, Mitra Mohtarami, Scott Cyphers, and Jim Glass. A Vector Space Approach for Aspect Based Sentiment Analysis. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 116–122, 2015.

[AMN+98] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[AP21] Sercan Ö. Arik and Tomas Pfister. TabNet: Attentive Interpretable Tabular Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6679–6687, May 2021.

[AR15] Alexandr Andoni and Ilya Razenshteyn. Optimal Data-Dependent Hashing for Approximate Near Neighbors. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*, pages 793–801, 2015.

[Ass08] Association for Logic, Language and Information. *ESSLLI 2008 Workshop on Distributional Lexical Semantics*, 2008.

[AZLY19] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. code2vec: Learning Distributed Representations of Code. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, 2019.

[Bak18] Amir Bakarov. A Survey of Word Embeddings Evaluation Methods. *CoRR*, abs/1801.09536, 2018.

[BAMK16] Danushka Bollegala, Mohammed Alsuhaibani, Takanori Maehara, and Kenichi Kawarabayashi. Joint Word Representation Learning using a Corpus and a Semantic Lexicon. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, pages 2690–2696. AAAI Press, 2016.

[BBBT12] Elia Bruni, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. Distributional Semantics in Technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 136–145. ACL, 2012.

[BBN19] Dan Brickley, Matthew Burgess, and Natasha Noy. Google Dataset Search: Building a Search Engine for Datasets in an Open Web Ecosystem. In *The World Wide Web Conference*, pages 1365–1375. ACM, 2019.

[BBS17] Rajesh Bordawekar, Bortik Bandyopadhyay, and Oded Shmueli. Cognitive Database: A Step towards Endowing Relational Databases with Artificial Intelligence Capabilities. *CoRR*, abs/1712.07199, 2017.

[BCZ+16] Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam Tauman Kalai. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. In *Advances in Neural Information Processing Systems*, pages 4349–4357. Curran Associates, Inc., 2016.

[BDC20] Rishi Bommasani, Kelly Davis, and Claire Cardie. Interpreting Pretrained Contextualized Representations via Reductions to Static Embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4758–4781. ACL, 2020.

[BDK14] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! A Systematic Comparison of Context-Counting vs. Context-Predicting Semantic Vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247. ACL, 2014.

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.

[Ben75] Jon Louis Bentley. Multidimensional Binary Search Trees used for Associative Searching. *Communications of the ACM*, 18(9):509–517, 1975.

[BG19] Yonatan Belinkov and James Glass. Analysis Methods in Neural Language Processing: A Survey. *Transactions of the Association for Computational Linguistics*, 7:49–72, 2019.

[BGJM17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[BGRS99] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "Nearest Neighbor" Meaningful? In *Proceedings of the 7th International Conference on Database Theory*, pages 217–235. Springer, 1999.

[BK16] Oren Barkan and Noam Koenigstein. Item2vec: Neural Item Embedding for Collaborative Filtering. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2016.

[BKK96] Stefan Berchtold, Daniel A Keim, and Hans-Peter Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 28–39. Morgan Kaufmann Publishers Inc., 1996.

[BKR+16] Miroslav Batchkarov, Thomas Kober, Jeremy Reffin, Julie Weeds, and David Weir. A Critique of Word Similarity as a Method for Evaluating Distributional Semantic Models. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 7–12. ACL, 2016.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331. ACM, 1990.

[BL00] Vanda Broughton and Heather Lane. Classification Schemes Revisited: Applications to Web Indexing and Searching. *Journal of internet cataloging*, 2(3-4):143–155, 2000.

[BL12] Artem Babenko and Victor Lempitsky. The Inverted Multi-Index. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3069–3076. IEEE, 2012.

[BN01] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Advances in Neural Information Processing Systems*, volume 14, pages 585–591. MIT Press, 2001.

[BS17] Rajesh Bordawekar and Oded Shmueli. Using Word Embedding to Enable Semantic Queries in Relational Databases. In *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning*, pages 1–4, 2017.

[BS20] Ursin Brunner and Kurt Stockinger. Entity Matching with Transformer Architectures - A Step Forward in Data Integration. In *International Conference on Extending Database Technology, Copenhagen, 30 March-2 April 2020*, pages 463–473. OpenProceedings, 2020.

[BSS+18] Felix Biessmann, David Salinas, Sebastian Schelter, Philipp Schmidt, and Dustin Lange. Deep Learning for Missing Value Imputation in Tables with Non-Numerical Data. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 2017–2025. ACM, 2018.

[BUGD+13] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-Relational Data. In *Advances in Neural Information Processing Systems*, pages 2787–2795. Curran Associates, Inc., 2013.

[BZH+21] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. BinaryBERT: Pushing the Limit of BERT Quantization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4334–4348. ACL, 2021.

[CCN16] J. Camacho-Collados and R. Navigli. Find the Word that does not belong: A Framework for an Intrinsic Evaluation of Word Vector Representations. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 43–50. ACL, 2016.

[CDA+17] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Cross-Lingual Focused Evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14. ACL, 2017.

[CH90] Kenneth Ward Church and Patrick Hanks. Word Association Norms, Mutual Information, and Lexicography. *Computational Linguistics*, 16(1):22–29, 1990.

[Cha02] Moses S Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388. ACM, 2002.

[CHZ+08] Michael J Cafarella, Alon Y Halevy, Yang Zhang, Daisy Zhe Wang, and Eugene Wu. Uncovering the Relational Web. In *Proceedings of the 11th International Workshop on Web and Databases*, 2008.

[CKS+17] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680. ACL, 2017.

[CLK+16] Fernando Chirigati, Jialu Liu, Flip Korn, You Wu, Cong Yu, and Hao Zhang. Knowledge Exploration Using Tables on the Web. *Proceedings of the VLDB Endowment*, 10(3):193–204, 2016.

[CLLM20] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *International Conference on Learning Representations*, 2020.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2009.

[CPT20] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1335–1349. ACM, 2020.

[CTT00] Hsin-Hsi Chen, Shih-Chung Tsai, and Jin-He Tsai. Mining Tables from Large Scale HTML Texts. In *COLING: The 18th International Conference on Computational Linguistics*, pages 166–172. Morgan Kaufmann, 2000.

[DCES04] Souripriya Das, Eugene Inseok Chong, George Eadon, and Jaannathan Srinivasan. Supporting Ontology-Based Semantic Matching in RDBMS. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, pages 1054–1065. VLDB Endowment, 2004.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. ACL, 2019.

[DDF+90] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-Sensitive Hashing Scheme Based on p-stable Distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*, pages 253–262. ACM, 2004.

[DM01] Inderjit S Dhillon and Dharmendra S Modha. Concept Decompositions for Large Sparse Text Data Using Clustering. *Machine Learning*, 42(1-2):143–175, 2001.

[Doz16] Timothy Dozat. Incorporating Nesterov Momentum into Adam. 2016.

[DSL+20] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. TURL: Table Understanding through Representation Learning. *Proceedings of the VLDB Endowment*, 14(3):307–319, 2020.

[DTXO21] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity-Based Approach. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 456–467. IEEE, 2021.

[EAT19] Sepehr Eghbali, Hassan Ashtiani, and Ladan Tahvildari. Online Nearest Neighbor Search using Hamming Weight Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(7):1729–1740, 2019.

[EBH+15] Julian Eberius, Katrin Braunschweig, Markus Hentsch, Maik Thiele, Ahmad Ahmadov, and Wolfgang Lehner. Building the Dresden Web Table Corpus: A Classification Approach. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, pages 41–50. IEEE, 2015.

[Edw21] Chris Edwards. The Best of NLP. *Communications of the ACM*, 64(4):9–11, 2021.

[Eis19] Jacob Eisenstein. *Introduction to Natural Language Processing*. MIT Press, 2019.

[EKS+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[ETBL15] Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. Top-K Entity Augmentation Using Consistent Set Covering. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*, pages 1–12. ACM, 2015.

[ETJ+18] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. Distributed Representations of Tuples for Entity Resolution. *Proceedings of the VLDB Endowment*, pages 1454–1467, 2018.

[FB74] Raphael A. Finkel and Jon Louis Bentley. Quad Trees a Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4(1):1–9, 1974.

[FBF77] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[FDJ+15] Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. Retrofitting Word Vectors to Semantic Lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1606–1615. ACL, 2015.

[FGM+01] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing Search in Context: The Concept Revisited. In *Proceedings of the 10th International Conference on World Wide Web*, pages 406–414. ACM, 2001.

[FMQ+18] Raul Castro Fernandez, Essam Mansour, Abdulhakim A Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. Seeping Semantics: Linking Datasets using Word Embeddings for Data Discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 989–1000. IEEE, 2018.

[FN75] Keinosuke Fukunaga and Patrenahalli M. Narendra. A Branch and Bound Algorithm for Computing k-Nearest Neighbors. *IEEE Transactions on Computers*, 24:750–753, 1975.

[GAKS14] Ivan Giangreco, Ihab Al Kabary, and Heiko Schuldt. ADAM - A Database and Information Retrieval System for Big Multimedia Collections. In *2014 IEEE International Congress on Big Data*, pages 406–413. IEEE, 2014.

[Gam19] Pablo Gamallo. Using the Outlier Detection Task to Evaluate Distributional Semantic Models. *Machine Learning and Knowledge Extraction*, pages 211–223, 2019.

[GAS16] Josu Goikoetxea, Eneko Agirre, and Aitor Soroa. Single or Multiple? Combining Word Representations Independently Learned from Text and WordNet. In *Thirtieth AAAI Conference on Artificial Intelligence*, pages 2608–2614. AAAI Press, 2016.

[GBH18] Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic Neural Networks. In *Advances in Neural Information Processing Systems*, pages 5350–5360. Curran Associates, Inc., 2018.

[GBL14] Bin Gao, Jiang Bian, and Tie-Yan Liu. WordRep: A Benchmark for Research on Learning Word Representations. *CoRR*, abs/1407.1640, 2014.

[GBM16] Ramanathan V Guha, Dan Brickley, and Steve Macbeth. Schema.org: Evolution of Structured Data on the Web. *Communications of the ACM*, 59(2):44–51, 2016.

[GCS19] Li Gong, Josep M Crego, and Jean Senellart. Enhanced Transformer Model for Data-To-Text Generation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 148–156. ACL, 2019.

[GF18] Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[GFEC16] Sahar Ghannay, Benoit Favre, Yannick Esteve, and Nathalie Camelin. Word Embedding Evaluation and Combination. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 300–305. European Language Resources Association (ELRA), 2016.

[GGPS20] Majid Ghasemi-Gol, Jay Pujara, and Pedro Szekely. Learning Cell Embeddings for Understanding Table Layouts. *Knowledge and Information Systems*, pages 1–26, 2020.

[GGS18] Majid Ghasemi-Gol and Pedro Szekely. TabVec: Table Vectors for Classification of Web Tables. *arXiv preprint arXiv:1802.06290*, 2018.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[GJ21] Prakhar Gupta and Martin Jaggi. Obtaining Better Static Word Embeddings Using Contextual Embedding Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers)*, pages 5241–5253. ACL, 2021.

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

[Gol17] Yoav Goldberg. Neural Network Methods for Natural Language Processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[GOTL20] Michael Günther, Philipp Oehme, Maik Thiele, and Wolfgang Lehner. Learning from Textual Data in Database Systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 375–384. ACM, 2020.

[Gra84] Robert Gray. Vector Quantization. *IEEE ASSP Magazine*, 1(2):4–29, 1984.

[GRE+17] Anna Lisa Gentile, Petar Ristoski, Steffen Eckel, Dominique Ritze, and Heiko Paulheim. Entity Matching on Web Tables: a Table Embeddings Approach for Blocking. In *Proceedings of the 20th International Conference on Extending Database Technology (EDBT)*, pages 510–513. OpenProceedings, 2017.

[Gro20] Martin Grohe. word2vec, node2vec, graph2vec, x2vec: Towards a Theory of Vector Embeddings of Structured Data. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 1–16. ACM, 2020.

[GSL+20] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *International Conference on Machine Learning*, pages 3887–3896. PMLR, 2020.

[GSTL20a] Michael Günther, Paul Sikorski, Maik Thiele, and Wolfgang Lehner. FacetE, 2020. https://doi.org/10.34740/KAGGLE/DS/540160.

[GSTL20b] Michael Günther, Paul Sikorski, Maik Thiele, and Wolfgang Lehner. FacetE: Exploiting Web Tables for Domain-Specific Word Embedding Evaluation. In *Proceedings of the Workshop on Testing Database Systems*, pages 1–6. ACM, 2020.

[GTGL21] Michael Günther, Maik Thiele, Julius Gonsior, and Wolfgang Lehner. Pre-Trained Web Table Embeddings for Table Discovery. In *Fourth Workshop in Exploiting AI Techniques for Data Management*, pages 24–31. ACM, 2021.

[GTL19a] Michael Günther, Maik Thiele, and Wolfgang Lehner. Fast Approximated Nearest Neighbor Joins For Relational Database Systems. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 225–244. Gesellschaft für Informatik, 2019.

[GTL19b] Michael Günther, Maik Thiele, and Wolfgang Lehner. RETRO: Relation Retrofitting For In-Database Machine Learning on Textual Data. *arXiv preprint arXiv:1911.12674*, 2019.

[GTL20] Michael Günther, Maik Thiele, and Wolfgang Lehner. RETRO: Relation Retrofitting For In-Database Machine Learning on Textual Data. In *Proceedings of the 23rd International Conference on Extending Database Technology (EDBT)*, pages 411–414. OpenProceedings, 2020.

[GTLY19] Michael Günther, Maik Thiele, Wolfgang Lehner, and Zdravko Yanakiev. Explore FREDDY: Fast Word Embeddings in Database Systems. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 529–532. Gesellschaft für Informatik, 2019.

[GTNL20] Michael Günther, Maik Thiele, Erik Nikulski, and Wolfgang Lehner. RetroLive: Analysis of Relational Retrofitted Word Embeddings. In *Proceedings of the 23rd International Conference on Extending Database Technology (EDBT)*, pages 607–610. OpenProceedings, 2020.

[GTZ+20] Leilei Gan, Zhiyang Teng, Yue Zhang, Linchao Zhu, Fei Wu, and Yi Yang. SemGloVe: Semantic Co-occurrences for GloVe from BERT. *arXiv preprint arXiv:2012.15197*, 2020.

[Gün18] Michael Günther. FREDDY: Fast Word Embeddings in Database Systems. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1817–1819. ACM, 2018.

[Gut84] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57. ACM, 1984.

[GVH+16] Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. SimVerb-3500: A Large-Scale Evaluation Set of Verb Similarity. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2173–2182. ACL, 2016.

[Har54] Zellig S Harris. Distributional Structure. *Word*, 10(2-3):146–162, 1954.

[HAYSZ11] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, volume 22, pages 1312–1317. AAAI Press, 2011.

[HDGK12] Guy Halawi, Gideon Dror, Evgeniy Gabrilovich, and Yehuda Koren. Large-Scale Learning of Word Relatedness with Constraints. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge discovery and Data Mining*, pages 1406–1414. ACM, 2012.

[Her07] Susan C. Herring. A Faceted Classification Scheme for Computer-Mediated Discourse. *Language@Internet*, 4(1), 2007.

[HMR86] Geoffrey E Hinton, James L McClelland, and David E Rumelhart. Distributed Representations. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, pages 77–109. MIT Press, 1986.

[HNM+20] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Mueller, Francesco Piccinno, and Julian Eisenschlos. TaPas: Weakly Supervised Table Parsing via Pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333. ACL, 2020.

[HPR+21] Vinh Thinh Ho, Koninika Pal, Simon Razniewski, Klaus Berberich, and Gerhard Weikum. Extracting Contextualized Quantity Facts from Web Tables. In *Proceedings of the Web Conference 2021*, pages 4033–4042. ACM, 2021.

[HPW21] Vinh Thinh Ho, Koninika Pal, and Gerhard Weikum. QuTE: Answering Quantity Queries from Web Tables. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2740–2744. ACM, 2021.

[HRK15] Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating Semantic Models with (Genuine) Similarity Estimation. *Computational Linguistics*, 41(4):665–695, 2015.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[HYL17] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034. Curran Associates, Inc., 2017.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613. ACM, 1998.

[IM18] Masajiro Iwasaki and Daisuke Miyazaki. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. *arXiv preprint arXiv:1810.07355*, 2018.

[ITMI21] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. TABBIE: Pretrained Representations of Tabular Data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3446–3456. ACL, 2021.

[IU18] Tomoharu Iwata and Naonori Ueda. Unsupervised Object Matching for Relational Data. *arXiv preprint arXiv:1810.03770*, 2018.

[JDS11] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[KB15] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations (Poster)*, 2015.

[KB19] Masahiro Kaneko and Danushka Bollegala. Gender-Preserving Debiasing for Pre-trained Word Embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1641–1650. ACL, 2019.

[KDSG+16] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. Magellan: Toward Building Entity Matching Management Systems. *Proceedings of the VLDB Endowment*, 9(12):1197–1208, 2016.

[KHC15] Douwe Kiela, Felix Hill, and Stephen Clark. Specializing Word Embeddings for Similarity or Relatedness. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2044–2048. ACL, 2015.

[Koc20] Elvis Koci. *Layout Inference and Table Detection in Spreadsheet Documents*. PhD thesis, Technische Universität Dresden, Dresden; Polytechnic University of Catalonia, Barcelona, 2020.

[KSKW15] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From Word Embeddings to Document Distances. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 957–966. PMLR, 2015.

[KTR+19] Elvis Koci, Maik Thiele, Josephine Rehak, Oscar Romero, and Wolfgang Lehner. DECO: A Dataset of Annotated Spreadsheets for Layout and Table Recognition. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1280–1285. IEEE, 2019.

[KTRL19] Elvis Koci, Maik Thiele, Oscar Romero, and Wolfgang Lehner. A Genetic-Based Search for Adaptive Table Recognition in Spreadsheets. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1274–1279. IEEE, 2019.

[KTRML16] Elvis Koci, Maik Thiele, Óscar Romero Moral, and Wolfgang Lehner. A Machine Learning Approach for Layout Inference in Spreadsheets. In *IC3K 2016: Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management: Volume 1: KDIR*, pages 77–88. SciTePress, 2016.

[KW17] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations*. OpenReview.net, 2017.

[Lam18] Maximilian Lam. Word2bits - Quantized Word Vectors. *arXiv preprint arXiv:1803.05651*, 2018.

[LB19] Oliver Lehmberg and Christian Bizer. Synthesizing N-ary Relations from Web Tables. In *Proceedings of the 9th International Conference on Web Intelligence, Mining and Semantics*. ACM, 2019.

[LD97] Thomas K Landauer and Susan T Dumais. A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge. *Psychological Review*, 104(2):211, 1997.

[LDH+15] Ran Levy, Liat Ein Dor, Shay Hummel, Ruty Rinott, and Noam Slonim. TR9856: A Multi-Word Term Relatedness Benchmark. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 419–424. ACL, 2015.

[Lei76] Gottfried Wilhelm Leibniz. Leibniz-Handschriften zur Philosophie LH 4, 5, 8. http://digitale-sammlungen.gwlb.de/resolve?id=00068621, 1676.

[LG14a] Omer Levy and Yoav Goldberg. Linguistic Regularities in Sparse and Explicit Word Representations. In *Proceedings of the 18th Conference on Computational Natural Language Learning*, pages 171–180. ACL, 2014.

[LG14b] Omer Levy and Yoav Goldberg. Neural Word Embedding as Implicit Matrix Factorization. *Advances in Neural Information Processing Systems*, 27:2177–2185, 2014.

[LJW+07] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search. In *33rd International Conference on Very Large Data Bases, VLDB 2007*, pages 950–961. ACM, 2007.

[LLS+20a] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep Entity Matching with Pre-Trained Language Models. *Proceedings of the VLDB Endowment*, 14(1):50–60, 2020.

[LLS+20b] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep Wntity Matching with Pre-Trained Language Models. *arXiv preprint arXiv:2004.00584*, 2020.

[LMP18] Ben Lengerich, Andrew Maas, and Christopher Potts. Retrofitting Distributional Embeddings to Knowledge Graphs with Functional Relations. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2423–2436, Santa Fe, New Mexico, USA, August 2018. ACL.

[LNK07] David Liben-Nowell and Jon Kleinberg. The Link-Prediction Problem for Social Networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[LRMB16] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. A Large Public Corpus of Web Tables containing Time and Context Metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 75–76. ACM, 2016.

[LSM13] M.-T. Luong, R. Socher, and C. D. Manning. Better Word Representations with Recursive Neural Networks for Morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113. ACL, 2013.

[LWW13] Lipyeow Lim, Haixun Wang, and Min Wang. Semantic queries by example. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 347–358. OpenProceedings, 2013.

[Man15] Christopher D. Manning. Computational Linguistics and Deep Learning. *Computational Linguistics*, 41(4):701–707, 2015.

[MBXS17] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in Translation: Contextualized Word Vectors. In *Advances in Neural Information Processing Systems*, pages 6297–6308. Curran Associates, Inc., 2017.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, (ICLR), Workshop Track*, 2013.

[Mit79] Jürgen Mittelstrass. The philosopher's Conception of Mathesis Universalis from Descartes to Leibniz. *Annals of Science*, 36(6):593–610, 1979.

[ML14] Marius Muja and David G Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[MLR+18] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34. ACM, 2018.

[MLS13] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation. *arXiv preprint arXiv:1309.4168*, 2013.

[MNZ+18] Renée J Miller, Fatemeh Nargesian, Erkang Zhu, Christina Christodoulakis, Ken Q Pu, and Periklis Andritsos. Making Open Data Transparent: Data Discovery on Open Data. *IEEE Data Eng. Bull.*, 41(2):59–70, 2018.

[Mor66] Guy M Morton. *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. International Business Machines Company, 1966.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[MST+16] Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Lina M Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Counter-Fitting Word Vectors to Linguistic Constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–148. ACL, 2016.

[MY18] Yu A Malkov and Dmitry A Yashunin. Efficient and Robust Approximate Nearest Neighbor Search using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2018.

[MYZ13] Tomas Mikolov, Scott Wen-tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*, pages 746–751. ACL, 2013.

[Nav02] Gonzalo Navarro. Searching in Metric Spaces by Spatial Approximation. *The VLDB Journal*, 11(1):28–46, 2002.

[NCV+17] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning Distributed Representations of Graphs. In *Proceedings of the 13th International Workshop on Mining and Learning with Graphs (MLG)*, 2017.

[NK17] Maximilian Nickel and Douwe Kiela. Poincaré Embeddings for Learning Hierarchical Representations. In *Advances in Neural Information Processing Systems*, pages 6341–6350. Curran Associates, Inc., 2017.

[NK18] Maximillian Nickel and Douwe Kiela. Learning Continuous Hierarchies in the Lorentz Model of Hyperbolic Geometry. In *International Conference on Machine Learning*, pages 3779–3788. PMLR, 2018.

[NMCC16] Eric Nalisnick, Bhaskar Mitra, Nick Craswell, and Rich Caruana. mproving Document Ranking with Dual Word Embeddings. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 83–84. International World Wide Web Conferences Steering Committee, 2016.

[NØL18] Farhad Nooralahzadeh, Lilja Øvrelid, and Jan Tore Lønning. Evaluation of Domain-specific Word Embeddings using Knowledge Resources. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

[NPZ+20] Fatemeh Nargesian, Ken Q Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J Miller. Organizing Data Lakes for Navigation. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1939–1950. ACM, 2020.

[NZPM18] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. Table Union Search on Open Data. *Proceedings of the VLDB Endowment*, 11(7):813–825, 2018.

[OB16] Eng-Jon Ong and Miroslaw Bober. Improved Hamming Distance Search using Variable Length Substrings. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2000–2008. IEEE, 2016.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710. ACM, 2014.

[PB21] Ralph Peeters and Christian Bizer. Dual-Objective Fine-Tuning of BERT for Entity Matching. *Proceedings of the VLDB Endowment*, 14:1913–1921, 2021.

[PCFN06] Rodrigo Paredes, Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. Practical Construction of K-Nearest Neighbor Graphs in Metric Spaces. In *International Workshop on Experimental and Efficient Algorithms*, pages 85–97. Springer, 2006.

[PD91] Ruben Prieto-Diaz. Implementing Faceted Classification for Software Reuse. *Communications of the ACM*, 34(5):88–97, 1991.

[PHLM01] Gerald Penn, Jianying Hu, Hengbin Luo, and Ryan McDonald. Flexible Web Document Analysis for Delivery to Narrow-Bandwidth Devices. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pages 1074–1078. IEEE, 2001.

[PL02] Patrick Pantel and Dekang Lin. Discovering Word Senses from Text. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 613–619. ACM, 2002.

[PLH+18] Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. Large-Scale Hierarchical Text Classification with Recursively Regularized Deep Graph-CNN. In *Proceedings of the 2018 World Wide Web Conference*, pages 1063–1072. ACM, 2018.

[PNI+18] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. ACL, 2018.

[PNL+19] Matthew E Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A Smith. Knowledge Enhanced Contextual Word Representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 43–54. ACL, 2019.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. ACL, 2014.

[PWS19] Nina Poerner, Ulli Waltinger, and Hinrich Schütze. BERT is Not a Knowledge Base (Yet): Factual Knowledge vs. Name-Based Reasoning in Unsupervised QA. *arXiv preprint arXiv:1911.03681*, 2019.

[Ran39] Shiyali Ramamrita Ranganathan. *Colon Classification*. Madras Library Association, Madras, 1939.

[RG65] Herbert Rubenstein and John B Goodenough. Contextual Correlates of Synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

[RG19] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3973–3983. ACL, 2019.

[RK04] Ryan Rifkin and Aldebaro Klautau. In Defense of One-Vs-All Classification. *Journal of Machine Learning Research*, 5(Jan):101–141, 2004.

[RK19] Julian Risch and Ralf Krestel. Domain-specific Word Embeddings for Patent Classification. *Data Technologies and Applications*, 2019.

[RKV95] Nick Roussopoulos, Stephen Kelley, and Frederic Vincent. Nearest Neighbor Queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79. ACM, 1995.

[Ron14] Xin Rong. Word2Vec Parameter Learning Explained. *arXiv preprint arXiv:1411.2738*, 2014.

[RS10] Radim Rehurek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *In Proceedings of the LREC 2010 Workshop on new Challenges for NLP Frameworks*, 2010.

[RZLL16] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392. ACL, 2016.

[SAH08] Chanop Silpa-Anan and Richard I. Hartley. Optimised KD-Trees for Fast Image Descriptor Matching. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, pages 1–8. IEEE, 2008.

[SC16] Robert Speer and Joshua Chin. An Ensemble Method to Produce High-Quality Word Embeddings. *arXiv preprint arXiv:1604.01692*, 2016.

[Sha75] Michael Ian Shamos. Geometric complexity. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, pages 224–233. ACM, 1975.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[Sie15] Scharolta Katharina Sienčnik. Adapting word2vec to Named Entity Recognition. In *Proceedings of the 20th Nordic Conference of Computational Linguistics*, volume 109, pages 239–243. Linköping University Electronic Press, 2015.

[SLMJ15] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation Methods for Unsupervised Word Embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307. ACL, 2015.

[SLS18] Prathusha Kameswara Sarma, Yingyu Liang, and Bill Sethares. Domain Adapted Word Embeddings for Improved Sentiment Classification. In *Proceedings of the Workshop on Deep Learning Approaches for Low-Resource NLP*, pages 51–59. ACL, 2018.

[SMH+16] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. Table Cell Search for Question Answering. In *Proceedings of the 25th International Conference on World Wide Web*, pages 771–782. ACM, 2016.

[SN12] Mike Schuster and Kaisuke Nakajima. Japanese and Korean Voice Search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[SRF87] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 507–518. Morgan Kaufmann Publishers Inc., 1987.

[SWY75] Gerard Salton, Anita Wong, and Chung-Shu Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[SZ03] Josef Sivic and Andrew Zisserman. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Proceedings of the 9th IEEE International Conference on Computer Vision - Volume 2*, page 1470. IEEE, 2003.

[TDSL00] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[TFL+21] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Samuel Madden, and Mourad Ouzzani. RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation. *Proceedings of the VLDB Endowment*, 14(8):1254–1261, 2021.

[TGH19] Julien Tissier, Christophe Gravier, and Amaury Habrard. Near-Lossless Binarization of Word Embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7104–7111. AAAI Press, 2019.

[TML15] Andrew Trask, Phil Michalak, and John Liu. sense2vec - A Fast and Accurate Method for Word Sense Disambiguation in Neural Word Embeddings. *arXiv preprint arXiv:1511.06388*, 2015.

[TP10] Peter D Turney and Patrick Pantel. From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research*, 37:141–188, 2010.

[TWS20] T. P. Tanon, G. Weikum, and F. Suchanek. YAGO 4: A Reason-able Knowledge Base. In *European Semantic Web Conference*, pages 583–596. Springer, 2020.

[TYS+21] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Halevy. From natural language processing to neural databases. *Proceedings of the VLDB Endowment*, 14(6):1033–1039, 2021.

[VK14] Denny Vrandečić and Markus Krötzsch. Wikidata: A Free Collaborative Knowledge Base. *Communication of the ACM*, 57(10):78–85, 2014.

[VM15] Ivan Vulić and Marie-Francine Moens. Monolingual and Cross-Lingual Information Retrieval Models Based on (Bilingual) Word Embeddings. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 363–372. ACM, 2015.

[Voo94] Ellen M Voorhees. Query Expansion using Lexical-Semantic Relations. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 61–69. Springer, 1994.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, pages 5998–6008. Curran Associates, Inc., 2017.

[WDJ+21] Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. TUTA: Tree-based Transformers for Generally Structured Table Pretraining. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790. ACM, 2021.

[WH02] Yalin Wang and Jianying Hu. A Machine Learning Based Approach for Table Detection on the Web. In *Proceedings of the 11th International Conference on World Wide Web*, pages 242–250. ACM, 2002.

[WSC+16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*, 2016.

[WSL+21] Daheng Wang, Prashant Shiralkar, Colin Lockard, Binxuan Huang, Xin Luna Dong, and Meng Jiang. TCN: Table Convolutional Network for Web Table Interpretation. In *Proceedings of the Web Conference 2021*, pages 4020–4032. ACM, 2021.

[WSM+19] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355. ACL, 2019.

[WWW+20] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data. *Proceedings of the VLDB Endowment*, 13(12):3152–3165, 2020.

[WWZ+13] Jing Wang, Jingdong Wang, Gang Zeng, Rui Gan, Shipeng Li, and Baining Guo. Fast Neighborhood Graph Search using Cartesian Concatenation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2128–2135. IEEE, 2013.

[WYG+21] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, pages 2614–2627. ACM, 2021.

[WZFC14] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge Graph and Text Jointly Embedding. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1591–1601. ACL, 2014.

[XBB+14] Chang Xu, Yalong Bai, Jiang Bian, Bin Gao, Gang Wang, Xiaoguang Liu, and Tie-Yan Liu. RC-NET: A General Framework for Incorporating Knowledge into Word Representations. In *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*, pages 1219–1228. ACM, 2014.

[XFS14] Bai Xue, Chen Fu, and Zhan Shaobin. A Study on Sentiment Computing and Classification of Sina Weibo with Word2vec. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 358–363. IEEE, 2014.

[YD14] Mo Yu and Mark Dredze. Improving Lexical Embeddings with Semantic Knowledge. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 545–550. ACL, 2014.

[YHPC18] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent Trends in Deep Learning Based Natural Language Processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75, 2018.

[YLFW20] Wen Yang, Tao Li, Gai Fang, and Hong Wei. PASE: PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2241–2253. ACM, 2020.

[YLK10] Bin Yao, Feifei Li, and Piyush Kumar. K Nearest Neighbor Queries and kNN-Joins in Large Relational Databases (Almost) for Free. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 4–15. IEEE, 2010.

[YML19] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph Convolutional Networks for Text Classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7370–7377. AAAI Press, 2019.

[YNYR20] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426. ACL, 2020.

[YPS+20] Wenhao Yu, Wei Peng, Yu Shu, Qingkai Zeng, and Meng Jiang. Experimental Evidence Extraction System in Data Science with Hybrid Table Features and Ensemble Learning. In *Proceedings of The Web Conference 2020*, pages 951–961. ACM, 2020.

[ZBSC18] Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 93–104. ACL, 2018.

[ZD95] Justin Zobel and Philip Dart. Finding Approximate Matches in Large Lexicons. *Software: Practice and Experience*, 25(3):331–345, 1995.

[ZHL+19] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: Enhanced Language Representation with Informative Entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451. ACL, 2019.

[ZKBA15] Guido Zuccon, Bevan Koopman, Peter Bruza, and Leif Azzopardi. Integrating and Evaluating Neural Word Embeddings in Information Retrieval. In *Proceedings of the 20th Australasian Document Computing Symposium*. ACM, 2015.

[ZML+20] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. RepBERT: Contextualized Text Embeddings for First-Stage Retrieval. *arXiv preprint arXiv:2006.15498*, 2020.

[ZSW+19] Chaoqun Zhan, Maomeng Su, Chuangxian Wei, Xiaoqiang Peng, Liang Lin, Sheng Wang, Zhe Chen, Feifei Li, Yue Pan, Fang Zheng, et al. AnalyticDB: Real-time OLAP Database System at Alibaba Cloud. *Proceedings of the VLDB Endowment*, 12(12):2059–2070, 2019.

[ZWX15] X. Zhou, X. Wan, and J. Xiao. Representation Learning for Aspect Category Detection in Online Reviews. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 417–423. AAAI Press, 2015.

[ZYZZ18] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network Representation Learning: A Survey. *IEEE Transactions on Big Data*, 6(1):3–28, 2018.

[ZZB19] Li Zhang, Shuo Zhang, and Krisztian Balog. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1029–1032. ACM, 2019.

[ZZW+15] Huaping Zhong, Jianwen Zhang, Zhen Wang, Hai Wan, and Zheng Chen. Aligning Knowledge and Text Embeddings by Entity Descriptions. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 267–272. ACL, 2015.

LIST OF FIGURES

LIST OF TABLES

A

CONVEXITY OF RELATIONAL RETROFITTING

In the following, we prove the convexity of $\Psi(W)$ under certain hyperparameter configuration. We published this proof already in [GTL19b].

In the definition of the function $\Psi(W)$ in Section 5.2.2, W is procesed row-wise. Here, we consider each of its elements $w_{i,j}$ separately in function Ψ. This is possible since the quadratic Euclidean distances can be split in a sum of quadratic differences of coordinate values.

$$\begin{aligned}\Psi(W) = \sum_{d=1}^{D}\sum_{i=1}^{n}\Big[&\alpha_i(w_{i,d} - w'_{i,d})^2 + \beta_j\Psi_C(w_{i,d}, W) \\ &+ \Psi_R(w_{i,d}, W)\Big]\end{aligned} \tag{A.1}$$

$\Psi(W)$ can be split in $\Psi(W) = \hat{\Psi}(W) + \Psi_\beta(W) + \Psi_\gamma(W)$:

$$\begin{aligned}\hat{\Psi}(W) &= \sum_{d=1}^{D}\sum_{i=1}^{n}\Big[\alpha_i(w_{i,d} - w'_{i,d})^2 \\ &\quad - \sum_{r\in R}\Big[\sum_{k:(i,k)\in\widetilde{E_r}}\delta_i^r(w_{i,d} - w_{k,d})^2\Big]\Big] \\ \Psi_\beta(W) &= \sum_{d=1}^{D}\sum_{i=1}^{n}\Big(w_{i,d} - \frac{\sum_{j\in C(i)} w'_{j,d}}{|C(i)|}\Big)^2 \\ \Psi_\gamma(W) &= \sum_{d=1}^{D}\sum_{i=1}^{n}\sum_{r\in R}\Big[\sum_{j:(i,j)\in E_r}\gamma_i^r(w_{i,d} - w_{j,d})^2\Big]\end{aligned} \tag{A.2}$$

We utilize the fact, that a sum of convex functions is also convex. It is easy to see that $\Psi_\beta(W)$ and $\Psi_\gamma(W)$ are convex functions if all γ_i^r and β_i values are positive. $\Psi_\beta(W)$ is a simple D-dimensional quadratic function and thus convex. $\Psi_\gamma(W)$ consists of sums of squared differences which are themselves convex functions. Hence, in order to prove the convexity of $\Psi(W)$, it is sufficient to prove convexity for $\hat{\Psi}$. As a next step, we create the Hessian matrix of all second partial derivatives:

$$H = \begin{pmatrix} \frac{\partial^2\hat{\Psi}(W)}{\partial w_{1,1}^2} & \cdots & \frac{\partial^2\hat{\Psi}(W)}{\partial w_{1,1}\partial w_{n,D}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2\hat{\Psi}(W)}{\partial w_{n,D}\partial w_{1,1}} & \cdots & \frac{\partial^2\hat{\Psi}(W)}{\partial w_{n,D}^2} \end{pmatrix} \tag{A.3}$$

$$
\begin{aligned}
\frac{\partial^2 \hat{\Psi}(V)}{\partial w_{i,d}^2} =& 2\Big(\alpha - \sum_{r \in R} \Big[\sum_{\substack{k:(i,k) \\ \in \widetilde{E_r}}} (\delta_i^r + \delta_k^{\bar{r}}) \Big] \Big) \\
\frac{\partial^2 \hat{\Psi}(V)}{\partial w_{i,d}, \partial w_{k,d}} =& \sum_{r \in R} \phi(i,k,r) \;\; \textit{if } i \neq k \\
\phi(i,k,r) =& \begin{cases} 4\delta_i^r & (i,k) \in \widetilde{E_r} \\ 0 & \textit{otherwise} \end{cases} \\
\frac{\partial^2 \hat{\Psi}(V)}{\partial w_{i,d}, \partial w_{j,d'}} =& 0 \;\; \textit{if} \;\; d \neq d'
\end{aligned} \tag{A.4}
$$

$\hat{\Psi}(W)$ is convex if and only if the Hessian matrix is positive semi-definite. According to the parameter configuration defined in Section 5.2.3, the following condition holds:

$$
(i,k) \in \widetilde{E_r} \implies (k,i) \in \widetilde{E_{\bar{r}}} \wedge \delta_i^r = \delta_k^{\bar{r}} \tag{A.5}
$$

This leads to the following equivalence:

$$
\frac{\partial^2 \hat{\Psi}(V)}{\partial w_{i,d}, \partial w_{j,d'}} = \frac{\partial^2 \hat{\Psi}(V)}{\partial w_{j,d'}, \partial w_{i,d}} \tag{A.6}
$$

Subsequently, H is a symmetric matrix. An interesting matrix property to consider for a symmetric matrix (a_{ij}) is the diagonally dominance. A matrix is diagonally dominant, if in every row i the magnitude of the element on the diagonal $|a_{ii}|$ is greater or equal than the sum of magnitudes of the remaining elements $\sum_{j \neq i} |a_{ij}|$. If a matrix is diagonally dominant, it is also positive semi-definite. Therefore, the following is a sufficient condition to show that $\hat{\Psi}(W)$ is a convex function.

$$
\begin{aligned}
\forall i \in \{1, \dots n \times D\} : |h_{ij}| \geq & \sum_{j \neq i} |h_{ij}| \\
\forall i \in \{1, \dots n\}, d \in \{1, \dots, D\} : & \\
\Big| \frac{\partial^2 \hat{\Psi}(W)}{\partial w_{i,d}^2} \Big| \geq & \sum_{j \neq i} \Big| \frac{\partial^2 \hat{\Psi}(V)}{\partial w_{i,d}, \partial w_{j,d}} \Big| \\
2\Big| \alpha_i - \sum_{r \in R} \Big[\sum_{\substack{j:(i,j) \\ \in \widetilde{E_r}}} (\delta_i^r + \delta_j^r) \Big] \Big| \geq & \sum_{j \neq i} \Big| \sum_{r \in R} \phi(i,j,r) \Big| \\
\Big| \alpha_i - \sum_{r \in R} \Big[\sum_{\substack{j:(i,j) \\ \in \widetilde{E_r}}} 2\delta_i^r \Big] \Big| \geq & \sum_{r \in R} \Big| \sum_{\substack{j:(i,j) \\ \in \widetilde{E_r}}} 2\delta_i^r \Big|
\end{aligned} \tag{A.7}
$$

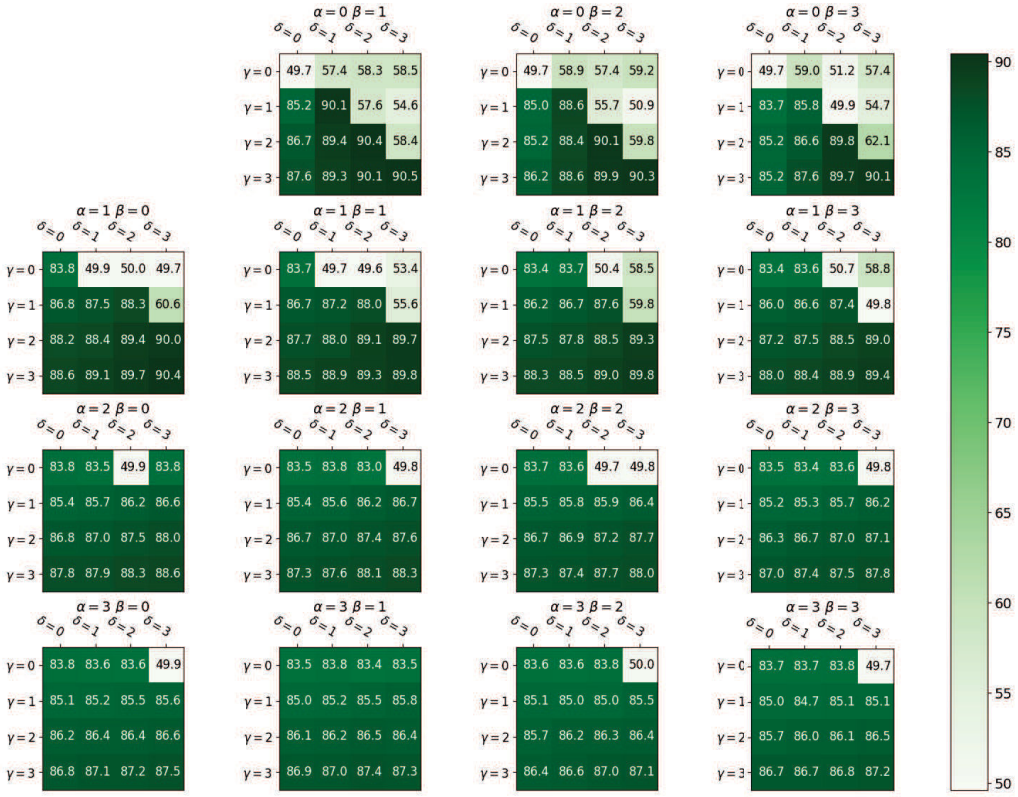
If we assume that for every i and r the values a_i and δ_i^r are positive, the solution of this inequality results in Equation (A.8):

$$
\alpha_i \geq 4 \sum_{r \in R} \Big[\sum_{\substack{j:(i,j) \\ \in \widetilde{E_r}}} \delta_i^r \Big] \tag{A.8}
$$

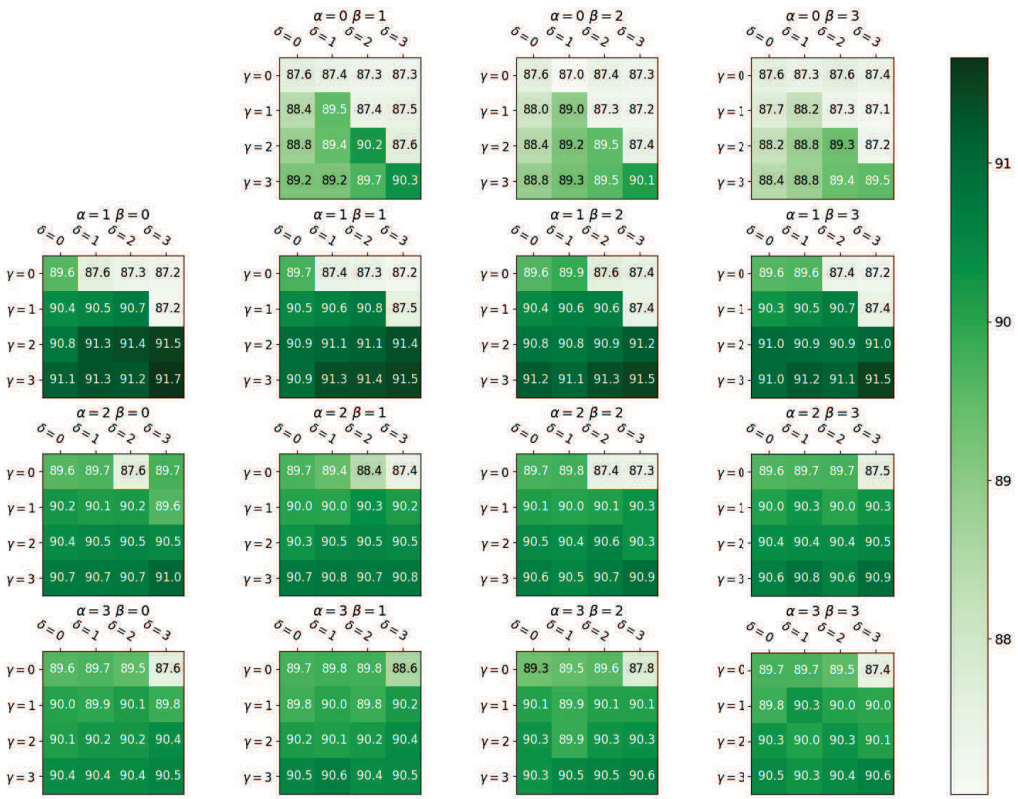
B

EVALUATION OF THE RELATIONAL RETROFITTING HYPERPARAMETERS

For the evaluation of the relational retrofitting hyperparameters α, β, γ, and δ, we performed a grid search for the binary classification of directors of the TMDB dataset (see Section 5.8). We applied the feed-forward neural network described in Section 5.4.3 on the relational retrofitted vectors and the embeddings obtained by the concatenation with node embeddings as described in Section 5.4.4. Figure B.1 shows the average accuracy values.



(a) Only Retrofitted Vectors



(b) Retrofitted Vectors combined with DeepWalk Embeddings

Figure B.1: Influence of Hyperparameters on Binary Classification for Relational Retrofitting with Ψ Function

CONFIRMATION

I confirm that I independently prepared the thesis and that I used only the references and auxiliary means indicated in the thesis.

Dresden, September 6, 2021