# SiLi Index: Data Structure for Fast Vector Space Searching

Ondřej Herman and Pavel Rychlý

Faculty of Informatics
Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
{xherman1,pary}@fi.muni.cz

**Abstract.** Nearest neighbor queries in high-dimensional spaces are expensive. In this article, we propose a method of building and querying a stand-alone data structure, SiLi (**Si**milarity **Li**st) Index, which supports approximating the results of k-NN queries in high-dimensional spaces, while using a significantly reduced amount of system memory and processor time compared to the usual brute-force search methods.

**Keywords:** word embeddings, vector space, semantic similarity

## 1 Introduction

### 1.1 Motivation

Vector space models have been central to the field of natural language processing for a long time, ranging from traditional sparse and high-dimensional bag-of-words document representations, where the vector space co-ordinates represent different words or phrases with tens of thousands of dimensions, to recent dense embeddings, which typically operate on hundreds of dimensions. The particular dimensions usually do not have clear interpretation, but as in the case of [2], the structure of the vector space has some interesting and useful properties.
Main operations of interest operating on dense vector spaces are the following:

1. **Pairwise similarity** – given two elements of the vector space, quantify their similarity.
2. ***k*-nearest neighbor queries** – given an element of the vector space, retrieve $k$ most similar elements.
3. **Analogy queries** – given three elements, $a$, $a^*$, and $b$, retrieve $k$ candidate elements $b^*$ which satisfy the following criterion: $a$ is to $a^*$ as $b$ is to $b^*$.

Evaluating pairwise similarity of two elements is cheap, as it is enough to retrieve the elements and then calculate the similarity. When the elements are stored in secondary storage, this means two seek operations and a single evaluation of similarity.

$k$-nearest neighbor query is significantly more demanding. The typical and widely deployed naïve algorithm calculates the pairwise similarity between the

A. Horák, P. Rychlý, A. Rambousek (eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2019, pp. 111–116, 2019. 

query vector and every other element of the vector space and then selects the top $k$ most similar elements. Processor performance is usually not the limiting factor in this case. The vectors to be compared need to be loaded from storage. Even the cost of transferring the model to the processor has significant cost.

For example, a fastText model calculated for the word attribute of the enTenTen 2013 corpus [3] has over 6 million distinct elements, which represent all corpus lexicon entries, which occur in the corpus at least five times. The sizes of the model for different vector lengths are:

| Dimension | Datatype | Total size | Note |
|---|---|---|---|
| 100 | float32 | 2.540 GiB | |
| 300 | float32 | 7.620 GiB | |
| 500 | float32 | 12.700 GiB | |
| 500 | float16 | 6.350 GiB | non-native datatype, slow |

A recent desktop computer (as of 2019) has approximately 50 GiB/s of available bandwidth between the processor and main memory, so even when the model of dimension 100 is used, the rate at which the queries can be evaluated is limited to 20 per second by memory transfers alone.

The performance of evaluating analogy queries is comparable, as an analogy query can be transformed to the $k$-nearest neighbor query in the following way: given the query $a$ is to $a^*$ as $b$ is to $b^*$ where $b^*$ is the vector we are looking for, a new vector $v$ is calculated as $v = b - a + a*$ and then a $k$-nearest neighbor query around $v$ is evaluated.

As can be seen, to obtain reasonable performance, the vector space elements to be searched need to be stored in main system memory – streaming the results from secondary storage would mean slowdown of two to three orders of magnitude.

## 2 Description

The main idea, on which the SiLi Index is based, is that it is not necessary to store the elements of the vector space themselves, but only the results of the $k$-nearest neighbor queries for every element.

### 2.1 Structure

The current version of the structure consists of three parts: the main record array, which stores nearest neighbors for every vector, an index, which provides mapping from numerical IDs to record array positions to enable fast lookups, and an external lexicon, which is a mapping between the lexicon elements and their IDs.

The lexicon is stored as a text file, with two lines of metadata. The first line describes the amount of elements contained in the model, while the second line is the dimensionality of the original vector space. The rest of the lines in the file are strings – lexicon elements, ordered by the frequency at which they appear in

the source corpus, with the IDs being implicitly encoded as the position of the lexicon element in the file:

| 6658558 |
|---|
| 100 |
| the |
| . |
| , |
| to |
| and |
| of |
| a |
| in |
| is |
| that |
| ⋮ |

The main data file consists of variable length records. Every record has a header, which contains two 4 byte values: the ID of the vector space element which is represented by the current vector, and the number $n$ of the most similar elements stored in the record. Then follow $n$ pairs of 4 byte integers, representing the $n$ most similar elements and their similarities, ordered by the similarity in descending order. The maximum similarity is represented as the value of $2^{20}$, while the minimum would be 0. For example, the record for the first element in the model build from enTenTen13 [3] would be:

| Address | ID | Number of elements |
|---|---|---|
| | Neighbor ID | Similarity |
| 0x000000 | 0 | 500 |
| 0x000008 | 0 | 1048575 |
| 0x000010 | 5 | 861673 |
| 0x000018 | 7 | 807157 |
| 0x000020 | 18 | 755537 |
| 0x000028 | 92 | 750653 |
| 0x000030 | 24 | 746858 |
| 0x000038 | 196 | 736494 |
| 0x000040 | 52 | 735677 |
| 0x000048 | 1 | 723016 |
| | ⋮ | ⋮ |

The third part of the structure maps the lexicon elements to the positions in the record array. The mapping from the lexicon IDs to positions is a flat binary table, where every element is an 8 byte long offset into the main data file. This way, only a single random access to the file is needed to locate the position in the main data file. The portion of this mapping corresponding to the portion of the lexicon as shown above would be:

| Address | Offset |
|---|---|
| 0x000000 | 0 |
| 0x000008 | 501 |
| 0x000010 | 1002 |
| 0x000018 | 1503 |
| 0x000020 | 2004 |
| 0x000028 | 2505 |
| 0x000030 | 3006 |
| 0x000038 | 3507 |
| 0x000040 | 4008 |
| 0x000048 | 4509 |
| ⋮ | ⋮ |

### 2.2 Generation

The SiLi Index consists of the most most similar elements for every word. Therefore, efficient calculation of similarities between all pairs of elements is essential. Common similarity measure is cosine similarity. If the vector space elements are normalized, this task reduces to matrix multiplication. The whole result is, however, very big, and would not fit in main memory. To work around this, we calculate similarities in batches, between a 100-element band and every element in the vector space. The resulting SiLi index for the enTenTen13 corpus consisting of 6658558 distinct elements was calculated in 75 m 38.690 s of wall-clock time.

### 2.3 *k*-nearest neighbor queries

Retrieving $k$ neighbors is trivial. First, the query string is translated to the ID using the lexicon file. Then, the corresponding offset is located in the offset table, which points to the location at which the actual record is stored in the main data file. If the lexicon is loaded in main memory, this means two seek operations, compared to the need to scan the whole model in the case of dense storage of word embeddings. The main memory requirements are much smaller.

### 2.4 Analogy Queries

Surprisingly, it is possible to evaluate analogy queries using only the information about nearest neighbors of the elements. The query $a$ is to $a^*$ as $b$ is to $b^*$, where $b$ is the element to be found, we need retrieve the list of nearest neighbours of $a$, $a^*$ and $b^*$, then calculate the intersection of these lists and evaluate the resulting similarity – we optimize

$$\arg\max_{b^*}(sim(b^*, b - a + a^*))$$

To evaluate similarity between two vectors, we use cosine similarity, defined as

$$sim(u, v) = cos(u, v) = \frac{u \cdot v}{\|u\| \cdot \|v\|}$$

Following the work of Mikolov et al. ([5]), we normalize the embedding vectors, so the similarity measure can be simplified to $sim(u, v) = u \cdot v$. Using basic algebra, the first equation can then be transformed ([6,4]) to the form

$$\arg\max_{b^*}(b^* \cdot b - b^* \cdot a + b^* \cdot a^*)$$

We store the cosine similarities of normalized vectors in the SiLi index, therefore $b^* \cdot b$, $b^* \cdot a$ and $b^* \cdot a^*$ can be extracted from the records for the elements $b$, $a$ and $a^*$ respectively. This way, many analogy queries can be evaluated, alleviating the need for storage of the complete embeddings.

### 2.5 Future Work

Currently, we store 500 nearest neighbors for every element in the vector space. This is likely unnecessary. Storing an adaptive amount of neighbors depending on the frequency of the word and its neighbor would likely result only in marginal loss of recall and accuracy of the model. Another interesting signal is the specific distribution of the neighbors of a specific word. However, we found that the distribution itself is not correlated with frequency of the lexicon elements (1), so more research on this topic would be necessary.

The headers in the main record array are not necessary, but currently are kept for completeness, as the record array can be used without the mapping table, or the mapping table could be recovered from it. Another source of inefficiency is the storage of the similarity in 4 bytes, even though the amount of useful information contained in this value is significantly lower. However, the current format is aligned and easily machine readable. The influence of different encodings with respect to performance of the model needs to be evaluated.

Perhaps the most significant improvement would be obtained by creating a new record type for rare lexicon elements, where only a very small amount of the representants would be stored, between 1 to 5. The queries would be carried through these representants. The result would not be exact for the low frequency lexicon elements anymore, but would yield result with lower and upper bounds on the similarity. Rare words usually do not have high-quality embeddings, so this might not cause significant issues. The drawback of this approach is however reduced performance, as more seeks to the underlying storage would need to be carried out.

Some other approaches employing locally sensitive hashing show promise, but we find them overly complex, such as the FALCONN library ([1]).

## 3 Conclusion

We presented a data structure, SiLi Index, and accompanying algorithms which enable efficient $k$-nearest neighbor query evaluation from data stored on secondary storage. The data structure can also support many important instances of analogy queries.

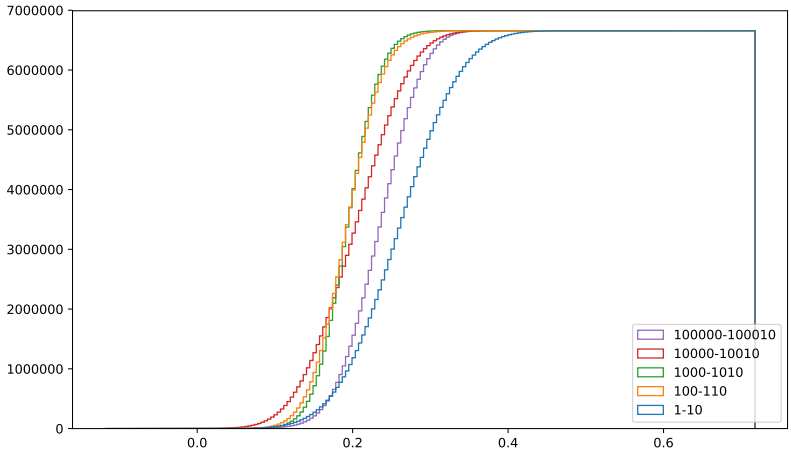


Fig. 1: Cumulative distribution of vector similarities for different rank bands for the word embedding model calculated using fastText on the enTenTen13 corpus.

**Acknowledgements** This work has been partly supported by the Ministry of Education of CR within the LINDAT-Clarin infrastructure LM2015071. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731015.

## References

1. Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., Schmidt, L.: Practical and optimal lsh for angular distance. In: Advances in Neural Information Processing Systems. pp. 1225–1233 (2015)
2. Grave, E., Mikolov, T., Joulin, A., Bojanowski, P.: Bag of tricks for efficient text classification. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL. pp. 3–7 (2017)
3. Jakubíček, M., Kilgarriff, A., Kovář, V., Rychlỳ, P., Suchomel, V.: The tenten corpus family. In: Proceedings of the 7th International Corpus Linguistics Conference CL. pp. 125–127 (2013)
4. Levy, O., Goldberg, Y.: Linguistic regularities in sparse and explicit word representations. In: Proceedings of the eighteenth conference on computational natural language learning. pp. 171–180 (2014)
5. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
6. Rychlỳ, P.: Evaluation of the sketch engine thesaurus on analogy queries. RASLAN 2016 Recent Advances in Slavonic Natural Language Processing p. 147 (2016)