Semantic representation of Slovak words

Šimon Horvát[1], Stanislav Krajči, and Ľubomír Antoni

Institute of Computer Science, Pavol Jozef Šafárik University in Košice, Slovakia,
simon.horvat@upjs.sk,

Abstract: The first problem one encounters when trying to apply analytical methods to text data is probably that of how to represent it in a way that is amenable to operations such as similarity, composition, etc. Recent methods for learning vector space representations of words have succeeded in capturing semantic using vector arithmetic, however, all of these methods need a lot of text data for representation learning.

In this paper, we focus on Slovak words representation that captures semantic information, but as the data source, we use a dictionary, since the public corpus of the required size is not available. The main idea is to represent information from the dictionary as a word network and learn a mapping of nodes to a low-dimensional space of features that maximize the likelihood of preserving network neighbourhoods of word nodes.

1 Introduction

Recently, the area of natural language processing (NLP) passed reform. Even this area did not escape the strong influence of the rise of neural networks. In most NLP classical tasks, such as text classification, machine translation, sentiment analysis, good results are achieved because of deep learning-based representation of fundamental building language components – words. Neural networks use large corpora for word representation learning [5][6].

However, in some languages, it is not possible to use this approach because it does not exist large amounts of unstructured text data in them. These issues can be well illustrated by Google translations in some not widely spoken languages (see Figure 1). The reason for poor translations is the absence of a large public source of text data.

In this paper, we propose our approach, which aims to obtain the semantic representation of words based on public dictionaries instead of the corpus. The main idea is to construct graph $G = (V, E, \phi)$ where vertices are words, which we want to get vector representation from and edges are relationships between words. It means the words that are closely related will be connected by an edge. The intensity of this relationship is expressed by weight – the real number from 0 to 1. As our source of data for building graph G, we used dictionaries. We will describe details in the section Data processing. We use `tf-idf` statistics for weighting our word network. After building graph G with mentioned properties, we apply well-known feature learning algorithm `Node2Vec` for networks. This method interprets nodes as vectors that are suitable for use as word vectors with semantic properties.

We focus on the process of Slovak words to increase the automated processing (NLP) of Slovak language, but it can be used for any other language. The paper is organized as follows: Section 2 – Related works, we present the basic definitions, and briefly describes methods from related works. Section 3 – Data processing, explores our data sources and their processing for the usage of our method. In Section 4 – Methods, We propose a novel approach, which produces dense vector representations of words with semantic properties. Some results are shown in section 5 – Experiments, and key takeaway ideas are discussed in the last section 6 – Conclusion.

2 Related works

Semantic vector space models of language represent each word with a real-valued vector. These representations are now commonly called word embeddings. The vectors can be used as features in a variety of applications as stated in the previous chapter. Distance between every two vectors should reflect how closely relate the meaning of the words, in ideal semantic vector space. The goal is to achieve an approximation of this vector space.

Word embeddings are commonly ([8][9][10]) categorized into two types, depending upon the strategies used to induce them. Methods that leverage local data (e.g. a word's context) are called prediction-based models and are generally reminiscent of neural language models. On the other hand, methods that use global information, generally corpus-wide statistics such as word counts and frequencies are called count-based models [4].

Both types of models have their advantages and disadvantages. However, a significant drawback of both approaches for not widely spoken language is the need for a large corpus. In the following sections, we show how can be this problem solved.

Prediction-based models. The idea of this approach is to learn word representations that aid in making predictions within local context windows (Figure 2). For example, Mikolov et al. [5] have introduced two models for learning embeddings, namely the continuous bag-of-words (CBOW) and skip-gram (SG) models (Figure 3). The main difference between CBOW and SG lies in the loss

Copyright ©2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-2718/paper12.pdf

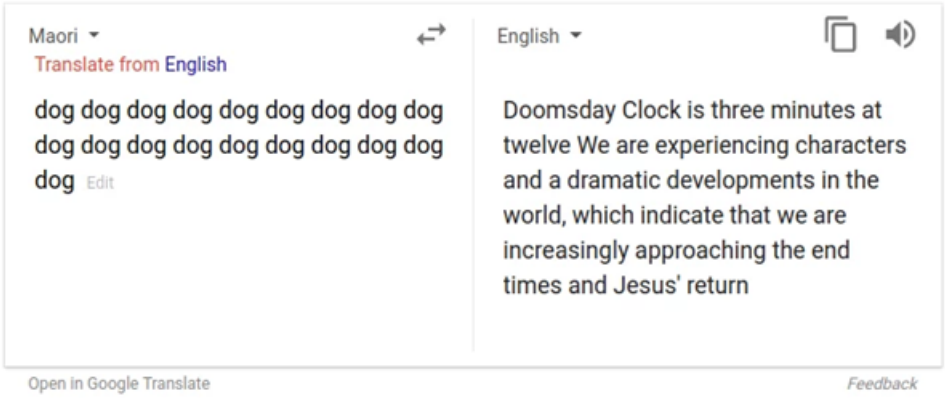

Figure 1: The languages that generate the strangest results – Somali, Hawaiian and Maori – have smaller bodies of translated text than more widely spoken languages like English or Chinese. As a result, it's possible that Google used religious texts like the Bible, which has been translated into many languages [20].

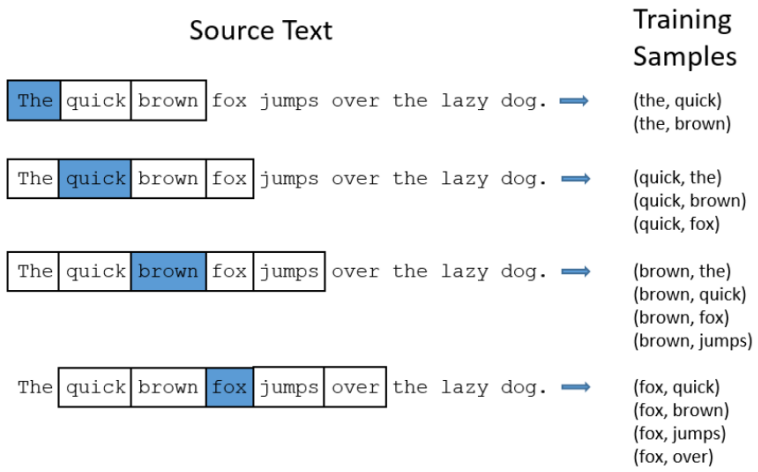

Figure 2: Local context windows. This figure shows some of the training samples (word pairs) we would take from the sentence *"The quick brown fox jumps over the lazy dog."* It used a small window size of 2 just for the example. The word highlighted in blue is the center word. The creation of a dataset for a neural network consists in such processing of each sentence in the corpus [19].

function used to update the model, while CBOW trains a model that aims to predict the center word based upon its context, in SG the roles are reversed, and the center word is, instead, used to predict each word appearing in its context.

Count-based models. These models are another way of producing word embeddings, not by training algorithms that predict the next word given its context but by leveraging word-context cooccurence counts globally in a corpus. These are very often represented (Turney and Pantel (2010) [15]) as word-context matrices. The earliest relevant example of leveraging word-context matrices to produce word embeddings is Latent Semantic Analysis (LSA) (Deerwester et al. (1990) [11]) where Singular value decomposition (SVD) is applied [4].

3 Data processing

As we already mentioned, we use dictionaries as our data source instead of corpora. First at all, we find web page, which contains dictionaries with public access for pulling data out of HTML (it is also possible to use dictionaries in text format). We parse two types of Dictionary:

1. Synonym dictionary [14],
2. classic dictionary [13] that contains a list of words and their meaning.

First, we establish some notation. Let $\mathbb{VOCAB}$ be a set of all words that we want to represent as a vector.

Let $\mathbb{S}$ represent the set of all synonym pairs achieved from the Synonym dictionary [14]. Set $\mathbb{S}$ contains pairs like

$$(vtipný, zábavný), (vtipný, smiešny), (rýchlo, chytro) \ldots$$

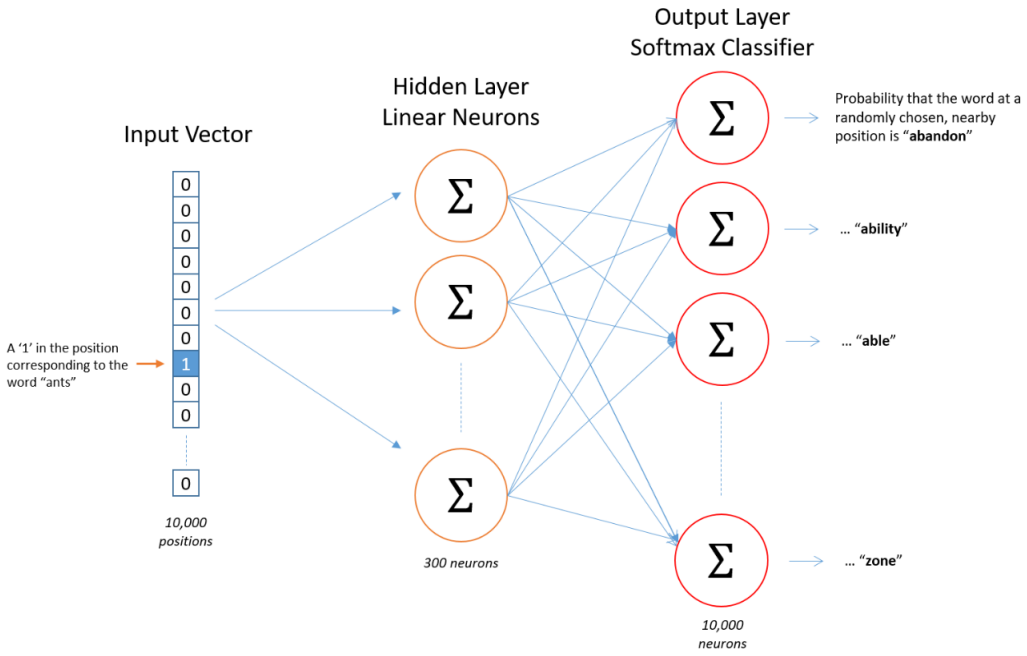

Figure 3: Skip-gram network architecture (the coded vocabulary is illustrative). As input, skip-gram expects center word in one-hot representation. The layer in the middle has the number of neurons as the desired dimension of word embeddings. The last layer tries to predict the probability distribution of the occurrence of words in the context of the center word. Metrix of weights between the middle and the last layer is word vectors [19].

It is important to remark that not every word from $\mathbb{VOCAB}$ has synonym pair. Let $\mathbb{L}$ represent the set of word pairs from the Dictionary [13].

We create these word pairs (w, l_i) as follow:

- For word w from $\mathbb{VOCAB}$, we find its definition from the Dictionary [13].
- Subsequently, we find the lemma of each word occurring in this definition of w. Let denote these lemmas as $l_1, l_2, \ldots, l_n$. For each word w from $\mathbb{VOCAB}$, there are pairs $(w, l_1), (w, l_2), \ldots, (w, l_n)$ in set $\mathbb{L}$. For instance, word *slnko* has definition: *"Nebeské teleso vysielajúce do vesmíru teplo a svetlo."* Based on that, we add to set $\mathbb{L}$ these pairs: *(slnko, nebeský), (slnko, teleso), (slnko, vysielajúce), (slnko, vesmír), (slnko, teplo), (slnko, svetlo).*

We used a rule-based tool for lemitization [16][17][18].

Let $G = (V, E, \phi)$ to be denoted by a directed graph where $V = \mathbb{VOCAB}$, edges $E = \mathbb{S} \cup \mathbb{L}$ and ϕ is the function that for each edge e from E assign real number $\phi(e)$. We will define function ϕ in section 4.1.

From now, our initial task of word representation learning is transformed into a graph-mining problem.

4 Methods

In this section, we present the `tf-idf` method and `Node2Vec` algorithm [1].

4.1 tf-idf

The notion `tf-idf` stands for term frequency-inverse document frequency, and the `tf-idf` weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the `tf-idf` weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. The `tf-idf` can be successfully used for stop-words filtering in various subject fields, including text summarization and classification. The `tf-idf` is the product of two statistics, term frequency and inverse document frequency [2][3].

Term frequency. Suppose we have a set of English text documents and wish to rank which document is most relevant to the query, *"the brown cow"*. A simple way to start out is by eliminating documents that do not contain all three words *"the"*, *"brown"*, and *"cow"*, but this still leaves many documents. To further distinguish them, we might count the number of times each term occurs in each document; the number of times a term occurs in a document is called its term frequency. In the case of the term frequency $\texttt{tf}(t, d)$, the simplest choice is to use the raw

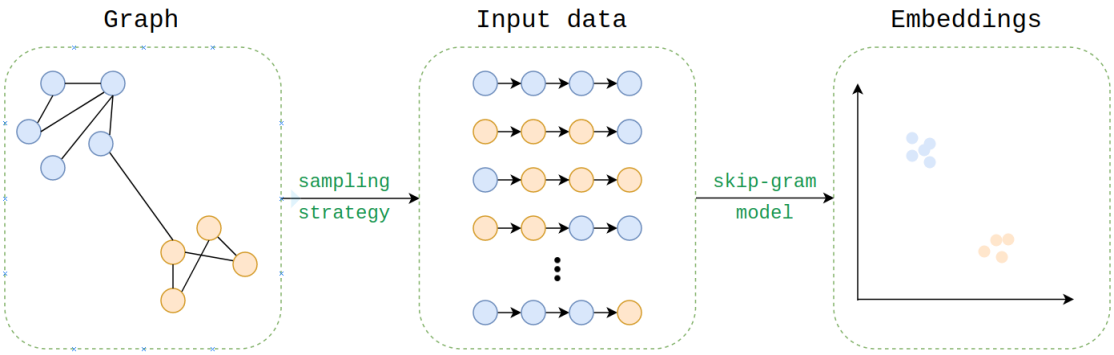

Figure 4: `Node2Vec` embedding process [21]

count of a term in a document, i.e., the number of times that term t occurs in document d.

Inverse document frequency. Because the term "the" is so common, term frequency will tend to incorrectly emphasize documents that happen to use the word "the" more frequently, without giving enough weight to the more meaningful terms "brown" and "cow". The term "the" is not a good keyword to distinguish relevant and non-relevant documents and terms, unlike the less-common words "brown" and "cow". Hence an inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely. So the inverse document frequency is a measure of how much information the word provides, i.e., if it's common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient):

$$\texttt{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{1}$$

with

- N: total number of documents in the corpus $N = |D|$
- $|\{d \in D : t \in d\}|$ number of documents where the term t appears.

Term frequency–Inverse document frequency. `tf-idf` is calculated as

$$\texttt{tf-idf}(t,d,D) = \texttt{tf}(t,d) \cdot \texttt{idf}(t,D)$$

A high weight in `tf-idf` is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms. Since the ratio inside the `idf`'s log function is always greater than or equal to 1, the value of `idf` (and `tf-idf`) is greater than or equal to 0. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the `idf` and `tf-idf` closer to 0.

tf-idf as a weight function. Let's consider word *železný* and its definition *"obsahujúci železo; majúci istý vzt'ah k železu"*. For Slovak readers, it is obvious not every word of the definition is related to the word *železný* in the same way. Some words are very important for the construction of definition *(obsahujúci, mat', istý)* but they are not related to the defined word. By the definition of our word network G, all lemmas will be joined with the word *železný* by an edge, but we can filter unrelated words by assigning them low weight.

- $\texttt{tf}(t,d)$ the number of times that word t occurs in definition d. For instance, $\texttt{tf}$*(železo, "obsahujúci železo; mat' istý vzt'ah železo")* = 2.
- $\texttt{idf}(t,D)$ is inverse document frequency defined as (1), where D is set of all definitions from the Dictionary,
  - N is total number of definitions,
  - and $|\{d \in D : t \in d\}|$ is number of definitions where the word t appears.

The definition implies that often appearing words in definitions (such as *"majúci"* or *"nejaký"*) have a low `idf` value. So the relationship between words w and l_i (lemma of i-th word that appears in definition d_w of word w) is given by value $\texttt{tf-idf}(w,l_i) = \texttt{tf}(l_i,d_w) \cdot \texttt{idf}(l_i,D)$.

`tf-idf` is our weight function if edge e join word w_1 and word w_2, where w_2 is the lemma of a word that appears in definition d_{w1} of word w, in other words, if e from $\mathbb{L}$. If edge e join synonyms words ($e \in \mathbb{S}$), the weight of e is 1 – a max weight value. If e belongs $\mathbb{L}$ but also e belongs $\mathbb{S}$, $\phi(e) = 1$.

$$\phi(e) = \phi(w_1,w_2) = \begin{cases} 1, & \text{if } e \in \mathbb{S} \\ \texttt{tf-idf}(w_1,w_2), & \text{if } e \in \mathbb{L} \\ 1, & \text{if } e \in \mathbb{S} \cap \mathbb{L} \end{cases}$$

4.2 Node2Vec

In previous sections, we have described building graph $G = (V,E,\phi)$ that captures the semantic relationships be-

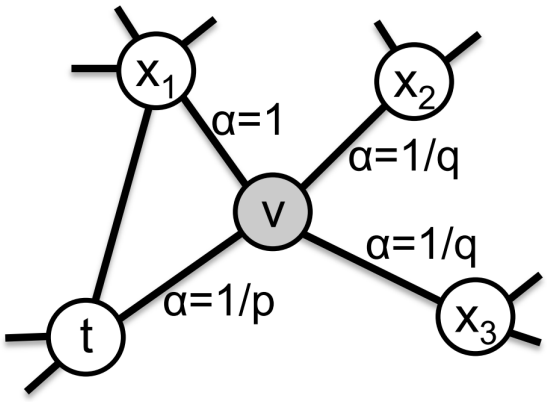

Figure 5: `Node2Vec` embedding process [1]

tween words. Finally, we need to obtain a vector representation of each node of a graph. We use the `Node2Vec` algorithm for this purpose [1]. The `Node2Vec` framework learns low-dimensional representations for nodes in a graph through the use of random walks. Given any graph, it can learn continuous feature representations for the nodes, which can then be used for various downstream machine learning tasks. `Node2Vec` follows the intuition that random walks through a graph can be treated like sentences in a corpus (sampling strategy). Each node in a graph is treated like an individual word, and a random walk is treated as a sentence. When we have a sufficiently large corpus obtained by random walks through the graph, the next step of the algorithm is to use the traditional embedding technique to obtain vector representation (see Figure 4), in the concrete, `Node2Vec` use mentioned skip-gram model (Figure 3). `Node2Vec` algorithm works in 2 steps: sampling strategy and feeding the skip-gram model. Since we already mention skip-gram model, we will focus on the sampling strategy.

`Node2Vec`'s sampling strategy, accepts four arguments:

- Number of walks n: Number of random walks to be generated from each node in the graph,
- walk length l: how many nodes are in each random walk,
- p: return hyperparameter,
- q: in-out hyperparameter.

The first two hyperparameters are self-explanatory. The algorithm for the random walk generation will go over each node in the graph and will generate n random walks, of length l.

Return parameter p controls the likelihood of immediately revisiting a node in the walk. Setting it to a high value ensures that we are less likely to sample an already visited node in the following two steps (unless the next node in the walk had no other neighbor). This strategy encourages moderate exploration and avoids 2-hop redundancy in sampling. On the other hand, if p is low, it would lead the walk to backtrack a step and this would keep the walk "local" close to the starting node u.

In-out parameter q allows the search to differentiate between "inward" and "outward" nodes. Going back to Figure 5, if $q > 1$, the random walk is biased towards nodes close to node t. Such walks obtain a local view of the underlying graph with respect to the start node in the walk and approximate BFS (Breadth First Search Traversal) behavior in the sense that our samples comprise of nodes within a small locality. In contrast, if $q < 1$, the walk is more inclined to visit nodes that are further away from the node t. Such behavior is reflective of DFS (Deph First Search Traversal) which encourages outward exploration. However, an essential difference here is that we achieve DFS-like exploration within the random walk framework. Hence, the sampled nodes are not at strictly increasing distances from a given source node u, but in turn, we benefit from tractable preprocessing and superior sampling efficiency of random walks [1].

For weighted graphs (our case), the weight of the edge has an impact on the probability of node visiting (higher weight - the higher probability of visiting).

5 Experiments

The word similarity measure is one of the most frequently used approaches to validate word vector representations. The word similarity evaluator correlates the distance between word vectors and human perceived semantic similarity. The goal is to measure how well the notion of human perceived similarity is captured by the word vector representations.

One commonly used evaluator is the cosine similarity defined by

$$\cos(w_x, w_y) = \frac{w_x \cdot w_y}{\|w_x\| \cdot \|w_y\|},$$

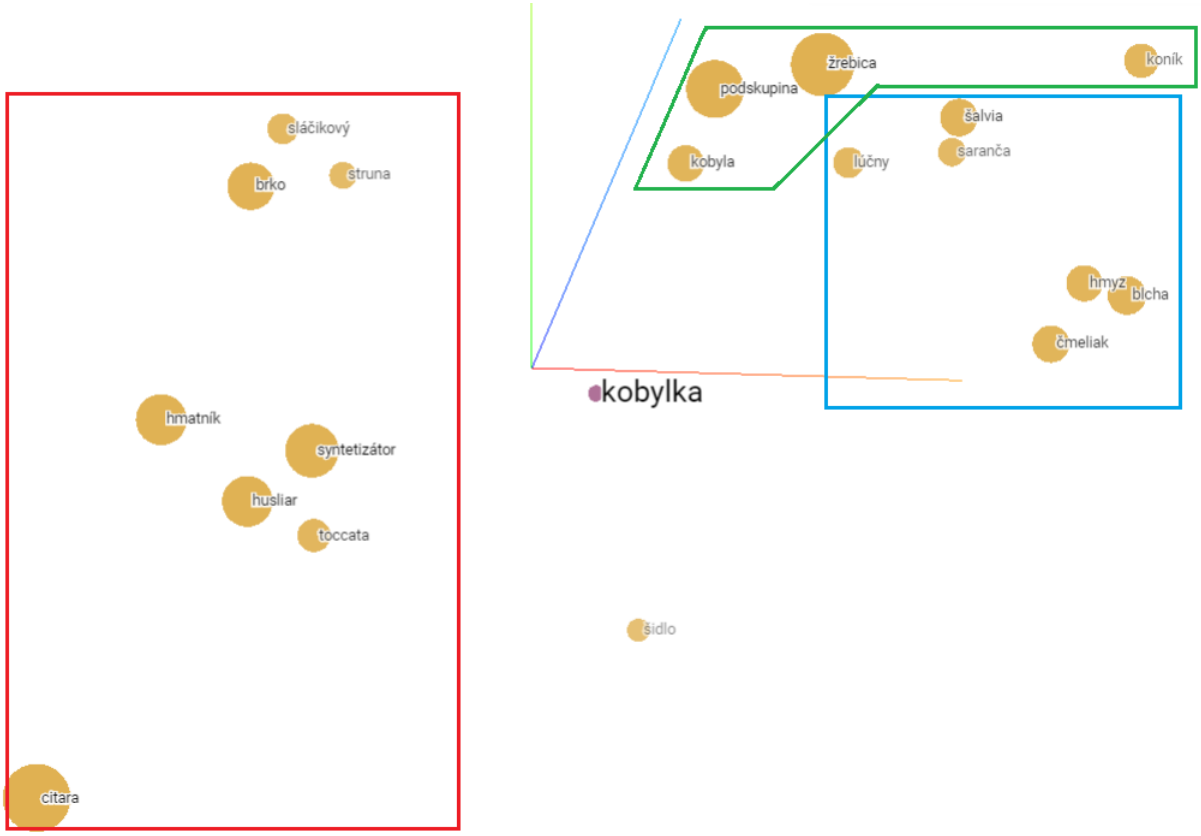

Figure 6: Word meanings communities of *kobylka* in word vector space.

where w_x and w_y are two word vectors and $\|w_x\|$ and $\|w_y\|$ are the L^2 norm.

This test computes the correlation between all vector dimensions, independent of their relevance for a given word pair or a semantic cluster. Many datasets are created for word similarity evaluation, unfortunately, there is no this kind of dataset for the Slovak language. In Table 1, we present the several words and list of their 20 nearest words, which are the results of our proposed model. We use cosine similarity as our distance metric and following setting of `Node2Vec` algorithm:

- Number of walks n: 20,
- walk length l: 100,
- p: 10,
- q: 1.

Several words in Table 1 have multiple meaning. For example, *kobylka* has 3 meanings:

1. miniature of a mare, female horse,
2. meadow jumping insect,
3. a string-supporting component of musical instruments.

Figure 6 shows that our word vector space captures the individual meanings of words by grouping words from one meaning into the community.

6 Conclusion

In this work, we offer a solution to the problem of a lack of text data for building word embedding. As a data source, we use a dictionary instead of a corpus. From the data, we have constructed the word network in which we transform each node into the vector space. In the section Experiments, we show that word vectors capture semantic information what is the main idea behind of word embedding. In addition, we have presented that vector space captures more senses for multiple meaning words.

As a possible extension of this work is to enrich our vector space with grammatical information too (vectors of adjectives will be closer to each other than vectors of adjective and verb). As we already mentioned, our graph contains only word lemmas, but it is also possible to add different shapes of a word into vector space.

In addition, we have presented that vector space is an appropriate representation for multiple meaning words.

radosť	**korenie**	**srdce**	**film**	**huba**	**tanec**	**ticho**	**kobylka**	**učiť**	**rýchlo**	**modra**	**verný**
lala	oregano	dušička	celuloidový	špongia	kozáčik	nehlučne	kobyla	vštepovať	šmihom	modrunký	nepredstieraný
zaľúbenie	bobuľka	dušinka	pornofilm	trúdnik	tarantela	nehlasne	sláčikový	zaúčať	friško	svetlobelasý	doslovný
jasot	čili	drahá	kinosála	michalka	sarabanda	bezhrmotne	saranča	vzdelávať	expresne	blankytný	úprimne
úľuba	fenikel	zlatko	diafilm	smrčok	špásy	nezvučne	lúčny	zaškoľovať	zvrtkom	temnobelasý	skalný
slasť	maggi	milá	krimi	tlama	tancovačka	bezhlasne	šalvia	školiť	prirýchlo	namodravý	mravčí
radostiplný	škorica	červeň	filmotéka	hľuzovka	odzemok	neslyšne	šidlo	školovať	gvaltom	kobaltový	faktický
potešenie	okoreniť	srdiečko	kinofilm	papuľa	foxtrot	tlmene	toccata	navykať	nanáhlo	nezábudkový	vytrvalý
potecha	korenička	chrobáčik	veľkofilm	čírovka	rumba	potíšku	koník	cvičiť	chvátavo	slabomodrý	pravdivý
pôžitok	ďumbier	miláčik	mikrofiš	mykológia	kratochvíľa	tíš	stradivárky	muštrovať	chvatom	temnomodrý	predstieraná
optimizmus	korenina	holúbok	indiánka	hubárčiť	menuet	bezhlučne	škripky	privykať	nanáhle	jasnobelasý	ozajstný
rozkoš	vanilka	sťah	vývojka	hubárka	kankán	tlmeno	prím	priúčať	chytro	bledobelasý	faksimile
blaženosť	pokoreniť	srdcovitý	porno	bedľa	radovánky	bezzvučne	tepovač	zvykať	prichytro	nebovomodrý	neprestávajúci
ujujú	majorán	srdciar	nitrocelulóza	hubár	mazúrka	neslyšateľne	mučidlo	vyučovať	zrýchlene	azúrový	opravdivý
ťjaj	sušený	kardiogram	kovbojka	peronospóra	twist	potíšky	husľový	memorovať	švihom	zafírovomodrý	vytrvanlivý
optimista	temian	centrum	videofilm	zvonovec	tamburína	nečujne	husle	osvojovať	skokmo	zafírový	naozajský
rozšťastnený	puškvorec	stred	predpremiéra	podpňovka	polonéza	pridusene	žrebica	trénovať	promptne	ultramarínový	úprimný
prešťastlivý	pamajorán	trombóza	premietareň	múčnatka	zábavka	nepočuteľne	viola	študírovať	habkom	nevädzový	neskreslený
ujú	aníz	kardiograf	pornohviezda	plávka	charleston	zmĺknuto	husliar	hlásať	úvalom	nevädzí	vtelený
zvesela	zázvor	drahý	detektívka	dubák	tango	tíško	struna	samouk	bleskurýchle	tuhobelasý	nefalšovaný

Table 1: The nearest 20 words to the first bold word.

References

[1] A. Grover and J. Leskovec: node2vec – Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016.

[2] J. Ramos: Using tf-idf to determine word relevance in document queries. In *ICML*, 2003.

[3] S. Robertson: Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*

[4] J. Pennington, R. Socher, and Ch. D. Manning: Glove – Global vectors for word representation. In *Conference on Empirical Methods on Natural Language Processing (EMNLP)*

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean: Efficient Estimation of Word Representations in Vector Space. In *ICLR Workshop Papers 2013a*

[6] G. Almeida, F. Xexéo: Word embeddings – a survey. arXiv preprint arXiv:1901.09069 (2019)

[7] T. Young, D. Hazarika, S. Poria, E. Cambria: Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine (2018).*

[8] M. Baroni, G. Dinu, and G. 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)*. Association for Computational Linguistics, June 2014.

[9] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and Ch. D. Manning: Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11. Association for Computational Linguistics 2011.

[10] S. Li, J. Zhu, and Ch. Miao: Agenerative word embedding model and its lowrank positive semidefinite solution 2015.

[11] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman: Indexing by latent semanticanalysis, 1990.

[12] K. Bollacker, D. Kurt, C. Lee: An Autonomous Web Agent for Automatic Retrieval and Identification of Interesting Publications. *Proceedings of the Second International Conference on Autonomous Agents. AGENTS '98. pp. 116–123.*

[13] Ľ. Štúr Institute of Linguistics of the Slovak Academy of Sciences (SAS): Krátky slovník slovenského jazyka 4. 2003 – kodifikačná príručka. Retrieved from `https://slovnik.aktuality.sk/`

[14] Ľ. Štúr Institute of Linguistics of the Slovak Academy of Sciences (SAS): Synonymický slovník slovenčiny, 2004.Retrieved from `https://slovnik.aktuality.sk/`

[15] P. D. Turney, P. Pantel: From Frequency to Meaning: Vector Space Models of Semantics. In *Journal of Artificial Intelligence Research*

[16] S. Krajči, R. Novotný, L. Turlíková, M. Laclavík: The tool Morphonary/Tvaroslovník: Using of word lemmatization in processing of documents in Slovak, in: *P. Návrat, D. Chudá (eds.), Proceedings Znalosti 2009, Vydavatel'stvo STU, Bratislava*, 2009, s. 119-130

[17] S. Krajči, R. Novotný – databáza tvarov slov slovenského jazyka, Informačné technológie – Aplikácie a teória, *zborník príspevkov z pracovného seminára ITAT*, 17.–21. september 2012, Monkova dolina (Slovensko), Košice, SAIS, Slovenská spoločnosť pre umelú inteligenciu, 2012, ISBN 9788097114411, s. 57–61

[18] S. Krajči, R. Novotný: Projekt Tvaroslovník – slovník všetkých tvarov všetkých slovenských slov, *Znalosti 2012, zborník príspevkov 11. ročníka konferencie: 14. - 16. október 2012, Mikulov (Česko), Praha, MATFYZPRESS, Vydavatelství MFF UK v Praze*, 2012, ISBN 9788073782207, s. 109–112

[19] C. McCormick: *Word2Vec Tutorial - The Skip-Gram Model*. Retrieved from `http://www.mccormickml.com`, 2016, April 19.

[20] J. Christian: Why Is Google Translate Spitting Out Sinister Religious Prophecies? Retrieved from `https://www.vice.com/en_us`, 2018.

[21] E. Cohen: node2vec: Embeddings for Graph Data. Retrieved from `https://towardsdatascience.com/`, 2018.