Kv2vec: A Distributed Representation Method for Key-value Pairs from Metadata Attributes

Chenxu Niu[1], Wei Zhang[1], Suren Byna[2], Yong Chen[1]
[1]Texas Tech University, [2]Lawrence Berkeley National Laboratory
{Chenxu.Niu,X-Spirit.Zhang}@ttu.edu, sbyna@lbl.gov, Yong.Chen@ttu.edu

2022 IEEE High Performance Extreme Computing Conference (HPEC) | 978-1-6654-9786-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPEC55821.2022.9926389

***Abstract*—Distributed representation methods for words have been developed for years, and numerous methods exist, such as word2vec, GloVe, and fastText. However, they are not designed for key-value pairs, which is an important data pattern and widely used in many scenarios. For example, metadata attributes of scientific files consist of a collection of key-value pairs. In this research, we propose `kv2vec`, a method that captures relationships between keys and values and represents key-value pairs in dense vectors. The fundamental idea of the `kv2vec` method is utilizing recurrent neural networks (RNNs) with long short-term memory (LSTM) hidden units to convert each key-value pair to a distributed vector representation. This new method overcomes the weaknesses of existing embedding models for representing key-value pairs as vectors. Moreover, it can be integrated into dataset search solutions through querying metadata attributes for self-describing file formats that are widely used in HPC systems. We evaluate the `kv2vec` method with multiple real-world datasets, and the results show that `kv2vec` outperforms existing models.**

***Index Terms*—Key-value pair, Distributed Representation Method, Metadata Search, Semantic Query**

I. Introduction

Many scientific applications using high-performance computing (HPC) systems nowadays often deal with massive amounts of experimental, observational, and simulation datasets, mostly in self-describing data formats, such as HDF5 [1], netCDF [2], ADIOS-BP [3]. The metadata in self-describing files provides detailed descriptive information about the internal data objects. Metadata can be accessed as a collection of *attributes* that are in the form of *key-value pairs* [4]. Each attribute key-value pair consists of a key representing the attribute name and a value representing the metadata value.

Given the increasing scale and complexity of scientific datasets, the search functionality has been instrumental in locating datasets of interests quickly. Most existing methods only apply *lexical (or literal) matching* of key-value pairs [5]–[10] to achieve search functionality. Capturing the features and representing *semantic vectors* for key-value pairs can, in many cases, improve the results of dataset search.

A natural thought is that we can apply existing distributed representation methods on key-value pairs to build semantic vectors to facilitate search. These methods have been proven to be a powerful tool applied in modern natural language processing (NLP) tasks. Numerous effective word embedding models have been proposed, such as `word2vec` [11], `GloVe` [12], and `fastText` [13]. Generally speaking, these approaches attempt to capture the semantics of a word by considering its relations with neighboring words in its context. Let us take `word2vec` as an example. It takes a large corpus of words as its input and generates the dense and low dimensional feature vector to represent each word. This feature has been proven very useful for finding the similar words and word-analogy (e.g., "tree is to leaf as flower is to petal"). In order to obtain the vectors of other types of contents (e.g., sentences, phrases, and documents), many researchers proposed methods like `paragraph2vec` [14], `entity2vec` [15], `image2vec` [16], and `item2vec` [17]. However, there are no methods designed for key-value pairs specifically. Each key and each value from a key-value pair can be considered as a phrase, but the relationships between keys and values matter equally. It is challenging to represent a key, a value and the potential relationship between them with a single vector.

In this study, we introduce `kv2vec` - a distributed representation method designed specifically for key-value pairs. `Kv2vec` aims to capture features from key-value pairs and represent them by semantic vectors. By introducing a recurrent neural network with long short-term memory, we are able to achieve the goal effectively. We have built a prototype of `kv2vec` and conducted extensive experimental tests on metadata attributes of real-world scientific datasets. The results indicate that, with the new `kv2vec` method, the chance of finding semantically relevant key-value pairs is much higher than the existing methods.

The contributions of this research study are:

- We identify limitations of existing methods in representing key-value pairs as semantic vectors.
- We introduce a new distributed representation method, `kv2vec`, for key-value pairs. Based on similarity measures of cosine distance between two vectors, we can measure the similarity of two key-value pairs and perform metadata queries on scientific datasets.
- We develop a prototype implementation of `kv2vec` and test out the implementation to validate our design. The evaluations show that, as compared to most existing solutions, `kv2vec` achieves desired accuracy while being efficient.

The rest of this paper is organized as follows. In Section II, we discuss the background and motivation of this work. Then

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

we present the design of the kv2vec method in Section III and describe the evaluation results in Section IV. We conclude this study and discuss future work in Section V.

II. Background and Motivation

In this section, we will introduce the distributed representation methods of general words and state-of-the-art distributed representation methods for key-value pairs.

A. Distributed Representations of Words

We briefly describe the concept of distributed representations of words in general. Distributed representations of words (a.k.a., word embeddings) are learned from a training dataset (for instance, Wikipedia corpus) in such a way that semantically related words are close to each other; i.e., the geometric relationship between words often also encode a semantic relationship between them. This embedding method seeks to map each word in a given vocabulary into a high dimensional vector with a fixed dimension, d (e.g., d could be 50, 100, 200 or 300). In other words, each word in a corpus is represented as a distribution of weights (positive or negative) across these dimensions by this method. Each word is represented by setting appropriate weights over multiple dimensions while each dimension of the vector contributes to the representation of many words with word embedding methods. That is why the vector representation is considered "distributed".

Word2vec is the first effective model to learn word associations from a large corpus of text with a neural network. It represents an algorithm that accepts text corpus as an input and outputs a vector representation for each word. There are two flavors of this algorithm, namely Skip-Gram and CBOW (Continuous Bag Of Words). Given a set of sentences the model loops on the words of each sentence and tries to use the current word w in order to predict its neighbors (i.e., its context). This approach is called "Skip-Gram". In CBOW, it uses each of these contexts to predict the current word, w. These models are all proven effective and fairly accurate in representing words. FastText is another effective word embedding method that is an extension of the word2vec model. It learns morphological details as well to achieve better accuracy. Furthermore, a method called GloVe applies the matrix factorization technique on generated word context matrix, in which a large matrix of co-occurrence is constructed for each word to show how frequently we see those words in the context of a large corpus.

All of the above models are proven to be effective in representing words with vectors. For instance, given a key-value pair "sensor name: laser", we can generate their corresponding vectors with word embedding models. Assume we choose word2vec in this example. Table I provides the vectors with 50 dimensions of the words (sensor, name and laser). Generally speaking, these approaches attempt to capture the potential semantics of a word by considering its relations with neighboring words in its context or the co-occurrence information.

TABLE I: Words and Represented Vectors

Words	Vectors
Sensor	(0.43, -0.32,..., 0.08)
Laser	(0.21, 0.75, ..., 0.26)
Name	(0.22, -0.14, ..., 0.16)

B. State-of-the-art Distributed Representation Methods for Key-value Pairs

All the above models are designed for words only, but our goal is generating vectors to represent key-value pairs. Two representative distributed representation methods, i.e., word averaging and sentence extension methods, exist for key-value pairs based on existing word embedding methods [11], [14].

In the word averaging method, for each key-value pair, we first break it into several individual words with a standard tokenizer. In the rest of the paper, we use "token" to represent words differentiated with a standard tokenizer. Tokenization is a way of separating a piece of text into smaller units. Here, tokens can be either words, characters, or sub-words. In key-value pairs of metadata attributes, all the tokens are words. For each word w of key-value pairs, we look up the pre-trained dictionary and retrieve the d-dimensional vector $v(w)$ from word2vec or GloVe.

In this naive approach, the vector representation for a key-value pair can be obtained by simply averaging the vectors of its words w. The vector representation of key-value pair kv is the concatenation of all vectors. For example, as shown in Figure 1, the vector of "sensor name: laser" is the average of these three individual vectors of "sensor", "name" and "laser".

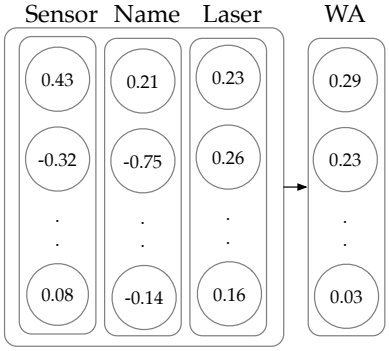

Figure 1: In the word averaging distributed representation method, the vector representation for a key-value pair is obtained by averaging the vectors of its individual words.

Apart from the word averaging method, another approach was designed to be applied for key-value pairs based on paragraph2vec [14], i.e., the sentence extension method. The goal of paragraph2vec (doc2vec) is to create a distributed representation of a paragraph or a document, regardless of its length. It is a generalization of the word2vec method. In the word averaging approach, we break each key-value pair into individual words. But in this approach, we do the opposite operation, which is sentence extension. We

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

insert words to rewrite the key-value pair into a meaningful sentence and generate distributed representation by applying `paragraph2vec` to this extended sentence. For example, the key-value pair "Sensor Name: Laser" can be extended to "the sensor name of the dataset is the Laser". This sentence is a normal sentence with meaningful context. The sentence would be used as an input for the `paragraph2vec` method and we will generate the corresponding vector.

C. Motivation

We have introduced these two representative existing methods to generate vectors for key-value pairs. However, there are shortcomings associated with these two methods. For the word averaging method, the well-known drawback is the ignorance of the word order. For instance, the vector generated by "Sensor name: Laser" is the same as the one from "Laser name: sensor". Apparently, they are completely different key-value pairs, but they are considered exactly the same if we apply the word averaging algorithm. In other words, the generated vectors would be the same if we switch the order of words or switch key and value part. It is critical to capture relationships between keys and values and generate more accurate distributed representations for key-value pairs. For the sentence extension approach, the accuracy is often limited since the key-value pair is not exactly the same as its extended sentence. If we select one syntax to extend a key-value pair, it may not be applicable or accurate for other key-value pairs. When the sentence/paragraph is long enough, `paragraph2vec` would be more accurate. As such, it is not a great fit to apply `paragraph2vec` on key-value pairs. To overcome these shortcomings, we need an alternative approach for computing semantic vectors for key-value pairs.

III. Methodology

In this section, we will first introduce a retrofitting technique to expand pre-trained dictionary. We will then introduce the design of `kv2vec` based on that.

A. Pre-trained Dictionary for Key-value Pairs

Before considering the potential relationships of each key-value pair, we need to explore how to construct embeddings for all words. To generate distributed representations for words, all existing approaches are based on one assumption: there exists a pre-trained dictionary for most (if not all) words of key-value pairs. In general cases, distributed representations of words are learned from a training dataset, such as the Common Crawl web corpus (840b tokens, 2.2M words) and Wikipedia 2014 (6B tokens, 400K words). The corpus determines the scope of one pre-trained dictionary. The number of words in these two corpus is very large. However, this assumption often does not hold for the words of key-value pairs in real-world scenarios. In the metadata attributes of scientific datasets, there are many domain-specific terminologies, which are not in the training corpus and pre-trained dictionary in most cases. Therefore, we need to introduce a method to enlarge the dictionary to include terms in key-value pairs. Below we will show how to apply retrofitting techniques to address this problem.

Many auxiliary semantic resources, such as PPDB [18], WordNet [19] and FrameNet [20], include a amount of semantic lexicons that can be applied to enlarge the dictionaries. In this study, we choose to use WordNet to perform the dictionary retrofitting on the pre-trained dictionary. WordNet is a large human-constructed semantic lexicon of English words. It groups those words into sets of synonyms called *synsets*, and provides short, general definitions, and records various semantic relations between them. This database is structured in a graph, particularly suitable for our problem because it explicitly relates concepts with semantically aligned relations such as hypernyms and hyponyms. If two words are related in WordNet, we will just refine their word embeddings/vectors to be similar. Based on the semantic lexicons, we can retrofit the pre-trained dictionaries to include terminologies that are specifically in the key-value pairs from metadata attributes.

Let $W = (k_1, k_2, \cdots, k_x, v_1, v_2, \cdots, v_y)$ be the set of individual words from a key-value pair. If some words "k_i" and "v_j" are not in the pre-trained dictionary, we apply the retrofitting technique to generate the vectors for these missing words. Let $U \subseteq W$ be the set of words with no word embeddings. First, we begin by looking up each word w in the set U. For example, if the word "precipitation" is missing in the dictionary, we can look up it in Wordnet and find out the synset: "rainfall", "downpour" and "rainwater". Then we collect the synset words and retrieve their vectors from the dictionary. These words are considered close to the missing word w and the semantic distance between each synset word and the missing word is small. To generate the vector for the missing word w, we calculate it based on all synset words. If we consider all of the synset words are equally close, then we set α as 1 and β as 0 (it is also the default setting). All of s_i are the synset words. If users want to differentiate the synset words, the number of α and β can be adjusted to achieve that. This process is repeated to generate vectors for all missing words in the set U.

$$v_w = \alpha \sum_{i=1}^{m} \frac{s_i}{m} + \beta \sum_{j=i+1}^{n} \frac{s_j}{n-m}$$

This retrofitting technique has a number of appealing properties. First, it is an efficient mechanism to learn the word embeddings of unknown words in many domains, especially for this study focusing on key-value pairs from metadata attributes to facilitate dataset search. Second, it provides an elegant mechanism to tune the word embeddings that is in sync with words in scientific domains. Moreover, domain experts can add more synsets of new terminologies based on their understandings. Finally, it allows us to efficiently enlarge the pre-trained dictionary without requiring large amounts of training datasets.

B. Distributed Representation Method for Key-value Pairs

The fundamental idea of our proposed `kv2vec` method is to use a neural network to compose the word vectors into a

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

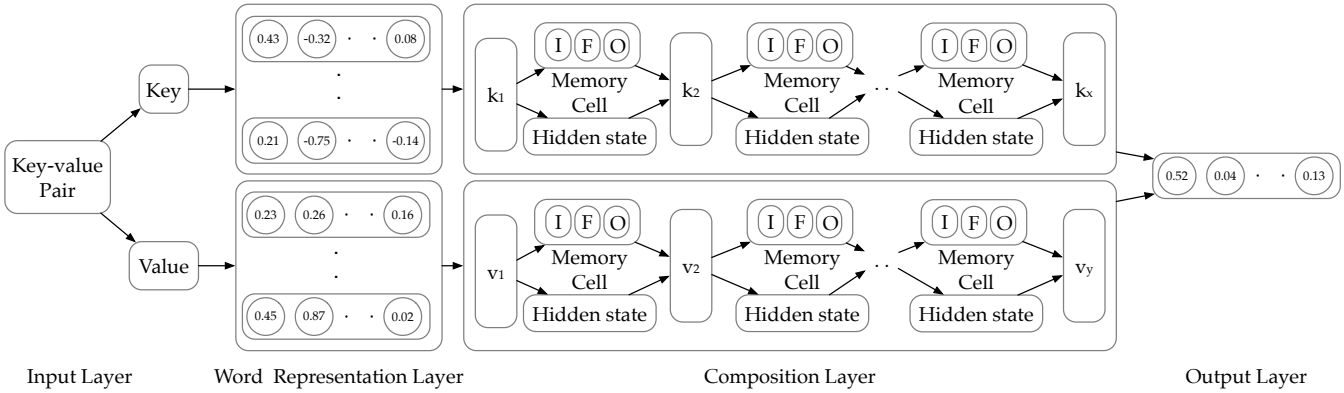

Figure 2: Overview of kv2vec workflow. The kv2vec utilizes RNN with LSTM to capture relationships between keys and values to generate distributed representations of semantic vectors. First, each key-value pair is split into key and value. Second, all individual words are extracted from the key and value, and these words are looked up from the pre-trained and retrofitted dictionary to construct vectors matching the words. Third, these vectors are fed into the composition layer with a long short-term memory network to produce the outputs, i.e., the kv2vec representation of semantic vectors.

key-value vector, instead of using word averaging or sentence extension algorithms. Considering the word order (and linguistic structure in general) and relationships between the keys and values could be important, as many key-value pairs contain multi-word content such as the description of a dataset. Many neural network architectures have been proposed to consider different types of linguistic structures, and the recurrent neural network structure is one such solution.

In our kv2vec method, we propose to use recurrent neural networks (RNN) with long short-term memory (LSTM) [21] hidden units, a.k.a. LSTM-RNNs. A recurrent neural network is a type of artificial network which uses sequential data or time series data. The concept behind RNNs is to make use of arbitrarily input data over long sequences, such that it repeats the same task to every element in the sequence and the output depends on the previous computation. Like normal "feed-forward" and convolutions neural networks (CNNs), recurrent neural networks utilize training data to learn. They are distinguished by their "memory" as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depends on the prior elements within the sequence. In this case, the applied network will still consider previous words when it deals with the following individual words of a "key" or a "value". That is why we can capture the relationships and unify the words of "key" as a meaningful phrase instead of unrelated words.

The overall kv2vec workflow is shown in Figure 2. There are four different layers in our design: an input layer, a word embedding layer, a composition layer and an output layer. First, each key-value pair is taken as the input and split into its "key" part and "value" part. After that, all individual words are extracted from the "key" and the "value" and these words are looked up from the pre-trained and retrofitted dictionary to construct all vectors matching the words. The next steps are taking all these vectors as inputs for the composition layer with a long short-term memory network (LSTM) to produce the outputs.

RNN with LSTM is capable of handling the vanishing gradient problem faced by RNN, which makes RNN more efficient. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a simple structure, such as a single tanh layer. In our scenario, the "value" part could be description of one object, which may be very long. As such, we apply LSTM in this recurrent neural network. There are three gates in the LSTM memory cell: input gate i_t, forget gate f_t and output gate o_t. All gates are generated by a sigmoid function over the ensemble of input x_t and the preceding hidden state h_{t-1}. The hidden state will be used to be compressed into the dense output vector which has the same dimensions with the vectors of words. In our case, in order to generate the hidden state at current step t, it first generates a temporary result q_t by a tanh non-linearity over the ensemble of input x_t and the preceding hidden state h_{t-1}. Then it combines this temporary result q_t with the previous word information h_{t-1} by input gate and forget gate f_t respectively to derive an updated internal information p_t. After that, it uses output gate o_t over this updated word information h_t to derive the final hidden state h_t. At the last step, it will combine all the words and convert them to one dense vector with the same dimension as the inputs. After the process of the composition layer, we can achieve the latest hidden state and the output vector to represent the entire key-value pair.

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

IV. Evaluation

In this section, we evaluate the efficacy and the performance of `kv2vec` over key-value pairs from different file formats with a series of experiments. We compare our method with the word averaging and the sentence extension methods. To better evaluate the functionality, we study the following aspects: pre-trained dictionary, the dictionary retrofitting efficiency, the overheads of vector creation process including storage consumption and computation complexity, as well as the efficacy of real-world search scenarios and applications.

A. Experimental Datasets

Many scientific datasets are stored in self-describing formats based on key-value pairs of metadata attributes. In this research, we pick NSIDC (National Snow and Ice Data Center) repository and datasets for the evaluation study. All these datasets are publicly available and are in HDF5 format, a typical example of self-describing format with metadata attributes in key-value pairs. We collected 1,987 files from the NSIDC repository for this study. The number of key-value pairs per file ranges from roughly 2,000 to 5,000. Additionally, to demonstrate the generality of our method, we choose Wikidata in JSON format and UDB (Ultimate Drill Book) files from Facebook for evaluation too.

B. Evaluation of Pre-trained Dictionary

In this series of experiments, we analyze the impact of the dictionary and evaluate which one is the best fit for the `kv2vec` method as the vectors of key-value pairs are based on a pre-trained dictionary from other models. We conducted experiments with three models: `word2vec`, `GloVe` and `fastText`. For the same model, the training datasets also matter. Common Crawl web corpus (840b tokens, 2.2M words) and Wikipedia 2014 (6B tokens, 400K words) are two training datasets for these models. After the training process is completed, we perform queries on key-value pairs from NSIDC metadata attributes and calculate the error rate of these queries. Table II shows the results of this series of tests. In general, there are only minor variations between different approaches. The `word2vec` model with Common Crawl web corpus has the least error rate. Therefore, we choose to use the pre-trained dictionary generated from the `word2vec` model with Common Crawl web corpus.

TABLE II: Error Rate of Query Results

Model	Training dataset	Error rate
`word2vec`	CCw	3.1%
`word2vec`	Wiki	4.5%
`GloVe`	CCw	3.3%
`GloVe`	Wiki	5.2%
`fastText`	CCw	3.1%
`fastText`	Wiki	4.9%

C. Evaluation of Retrofitted Dictionary

The goal of dictionary retrofitting is to enlarge the pre-trained dictionary for key-value pairs from scientific datasets or documents. With retrofitted dictionary, it is expected to achieve higher query hits for data in key-value pairs (α as 1 and β as 0). In this experiment, we extract all words from the key-value pairs from NSIDC datasets, match the dictionary with the word list and calculate the hit percentage. The accuracy of the generated vector is not the focus of this experiment, as we only measure the hit percentage of these queries. The results are plotted in Figure 3. The models have higher query hit rates when the dictionaries are retrofitted as can be observed from the figure (about 10% higher). Among these dictionaries, the pre-trained and retrofitted dictionary generated from the `word2vec` model with Common Crawl web corpus performed the best. In the following experiments, we choose to use the pre-trained and retrofitted dictionary generated by the `word2vec` model on the Common Crawl web corpus.

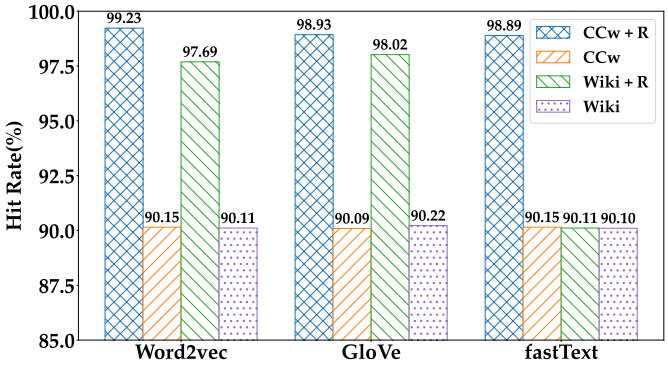

Figure 3: Query hit rate of different models with different dictionaries. For example, "CCw + R" means it is a Retrofitted dictionary on CCw datasets.

D. Analysis of Storage Consumption

Generating semantic vectors is the objective of `kv2vec`. From our preliminary observations, it will cost more storage space when the dimension of semantic vectors is increased. Additionally, `kv2vec` needs dictionary storage and storage to manage the results. In this experiment, we measure and study the storage consumption with different dimensions. The results are provided in Figure 4.

As shown from the figure, we analyzed the size of the storage of different dimensions over different datasets. For 50-element vectors, we need 171MB to store the vectors over NSIDC datasets. As expected, the size of required storage is proportional to the number of dimensions. For the UDB datasets, we need a much larger storage (1434MB) to hold 300-element vectors.

E. Analysis of Processing Time

We performed another experiment to evaluate the processing time. The processing time is related to the number of attributes

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

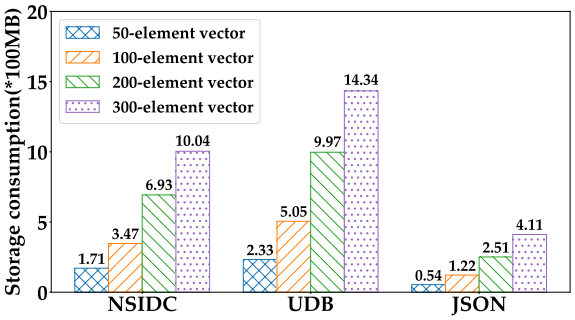

Figure 4: Storage consumption of the training process with different vector dimensions.

and the vector dimensions. As shown in Figure 5, it took 0.66 minutes to generate 50-element semantic vectors for the key-value pairs from NSIDC datasets, while it only took 0.12 minutes for JSON datasets. For UDB datasets, it took 0.98 minutes and 7.43 minutes for 50-element vectors and 300-element vectors, respectively. The reason is that the processing time is proportional to the number of key-value pairs and vector dimensions. Note that the training time reported in this figure is a one-time cost and the amortized cost over the number of search queries would be much lower.

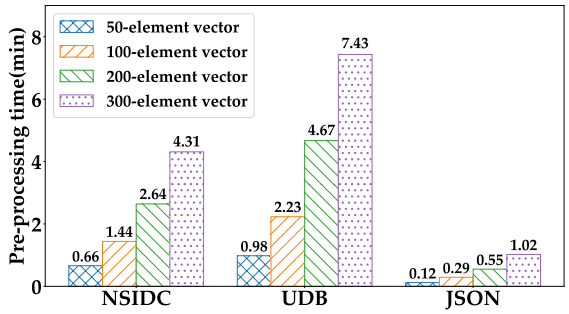

Figure 5: Computing/processing complexity analysis over different datasets with different vector dimensions.

F. Search Application

The `kv2vec` method can be utilized to perform queries on metadata attributes of files to locate datasets of interests. To evaluate the potential of this method for search scenarios, we collect a set of 10,000 key-value pairs randomly as query inputs. For each query, we calculate the cosine similarity between each attribute in the metadata and the query input and return the most similar one. To evaluate the quality of query results, we performed another experiment. In this experiment, we created a triplet of metadata attributes, with the first two attributes as returned results of the same query, whereas the third one is a randomly selected attribute from other metadata. The ability to identify which attribute is the randomly selected one shows the accuracy of the distributed representation methods. The rationale is that the results from the same query have smallest cosine distance, which is considered the ground truth in this test. We evaluated three methods and computed the cosine distances of the vectors generated from metadata attributes. The error rate of each method is the ratio of the number of incorrect results to the total number of triplets (i.e., 10,000). As shown in Table III, `kv2vec` had the smallest error rate for all dimensions compared against the other two methods for the NSIDC datasets. We also evaluated with two other datasets, JSON documents and UDB files. As shown in Table III, these results also confirmed the `kv2vec` method outperformed the word averaging and sentence extension methods, even though the error rate of these cases was higher than the tests with NSIDC datasets.

TABLE III: Error Rates of Different Methods

Model	Dim	Error Rate		
		NSIDC	UDB	JSON
Word averaging	50	17.3%	20.2%	9.9%
Word averaging	100	17.3%	20.2%	9.9%
Word averaging	200	17.3%	20.1%	9.9%
Word averaging	300	17.3%	20.1%	9.9%
Sentence Extension	50	15.6%	22.4%	9.2%
Sentence Extension	100	15.6%	22.4%	9.2%
Sentence Extension	200	15.6%	22.4%	9.2%
Sentence Extension	300	15.6%	22.4%	9.2%
`kv2vec`	50	3.1%	10.3%	8.1%
`kv2vec`	100	3.1%	10.3%	8.1%
`kv2vec`	200	3.1%	10.3%	8.1%
`kv2vec`	300	3.1%	10.3%	8.1%

V. Conclusion

In this paper, we have introduced a new distributed representation method called `kv2vec` to generate vectors for key-value pairs from metadata attributes. This method constructs vectors for key-value pairs and can be integrated into dataset search solutions through querying metadata attributes for self-describing file formats that are widely used in HPC systems. The `kv2vec` method applies a retrofitting technique to extend the pre-trained dictionary to include more words that are often missing in the training datasets. Based on that, this method utilizes a RNN with LSTM to capture relationships between keys and values, and then represent them by vectors. We have conducted extensive evaluations to compare our method with the state-of-the-art methods. The evaluation results show that the `kv2vec` method overcomes the limitation of existing solutions in the task of representing key-value pairs as vectors. We believe that such a method is useful for developing semantics understanding in dataset search solutions and for the development of advanced data management functionalities.

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

ACKNOWLEDGMENT

This research is supported in part by the National Science Foundation under grant CCF-1718336, CNS-1817094 and OAC-1835892. This work is supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] H. Group *et al.*, "HDF5 User's Guide," 2011.

[2] Wikipedia, "Network Common Data Format," https://en.wikipedia.org/wiki/NetCDF, 2018.

[3] J. F. Lofstead and et al., "Flexible IO and Integration for Scientific Codes through the Adaptable IO System (ADIOS)," in *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, ser. CLADE '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 15–24. [Online]. Available: https://doi.org/10.1145/1383529.1383533

[4] W. Zhang, S. Byna, C. Niu, and Y. Chen, "Exploring Metadata Search Essentials for Scientific Data Management," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. HiPC 2019. IEEE, 2019.

[5] W. Zhang, S. Byna, H. Tang, B. Williams, and Y. Chen, "MIQS: Metadata Indexing and Querying Service for Self-describing File Formats," in *SC Conference*, 2019, pp. 1–24.

[6] M. Lawson and J. Lofstead, "Using a Robust Metadata Management System to Accelerate Scientific Discovery at Extreme Scales," in *PDSW-DISCS*. IEEE, 2018, pp. 13–23.

[7] J. G. Institute, "The JGI Archive and Metadata Organizer(JAMO)," http://cs.lbl.gov/news-media/news/2013/new-metadata-organizer-streamlines-jgi-data-management, 2013.

[8] T. Craig E., E. Abdelilah, G. Dan *et al.*, "SPOT Suite," http://spot.nersc.gov/, 2013.

[20] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The Berkeley Framenet Project," in *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*, 1998.

[9] H. Sim, Y. Kim, S. S. Vazhkudai, G. R. Vallée, S. Lim, and A. R. Butt, "Tagit: An Integrated Indexing and Search Service for File Systems," in *SC conference*, 2017, pp. 5:1–5:12.

[10] H. Tang, S. Byna, B. Dong, J. Liu, and Q. Koziol, "SoMeta: Scalable Object-centric Metadata Management for High Performance Computing," in *CLUSTER Conference*. IEEE, 2017, pp. 359–369.

[11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and Their Compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[12] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *EMNLP Conference*, 2014, pp. 1532–1543.

[13] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext: Compressing Text Classification Models," *arXiv preprint arXiv:1612.03651*, 2016.

[14] Q. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *International conference on machine learning*. PMLR, 2014, pp. 1188–1196.

[15] R. Blanco, G. Ottaviano, and E. Meij, "Fast and Space-Efficient Entity Linking for Queries," in *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, 2015, pp. 179–188.

[16] D. Garcia-Gasulla, E. Ayguadé, J. Labarta, J. Béjar, U. Cortés, T. Suzumura, and R. Chen, "A Visual Embedding for the Unsupervised Extraction of Abstract Semantics," *Cognitive Systems Research*, vol. 42, pp. 73–81, 2017.

[17] O. Barkan and N. Koenigstein, "Item2vec: Neural Item Embedding for Collaborative Filtering," in *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2016, pp. 1–6.

[18] J. Ganitkevitch, B. Van Durme, and C. Callison-Burch, "PPDB: The Paraphrase Database," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 758–764.

[19] M. Faruqui, J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith, "Retrofitting Word Vectors to Semantic Lexicons," *arXiv preprint arXiv:1411.4166*, 2014.

[21] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

978-1-6654-9786-2/22/$31.00 ©2022 IEEE