General Model for Index Recommendation Based on Convolutional Neural Network

Yu Yan[1] and Hongzhi Wang[1,2](✉)

[1] Harbin Institute of Technology, Harbin, China
{yuyan0618,wangzh}@hit.edu.cn
[2] Peng Cheng Laboratory, Shenzhen, China

Abstract. With the advent of big data, the cost of index recommendation (IR) increases exponentially, and the portability of IR model becomes an urgent problem to be solved. In this paper, a fine-grained classification model based on multi-core convolution neural network (CNNIR) is proposed to implement the transferable IR model. Using the strong knowledge representation ability of convolution network, CNNIR achieves the effective knowledge representation from data and workload, which greatly improves the classification accuracy. In the test set, the accuracy of model classification reaches over 95%. CNNIR has good robustness which can perform well under a series of different learning rate settings. Through experiments on MongoDB, the indexes recommended by CNNIR is effective and transferable.

Keywords: Document · Database · Index recommendation · CNN · Class model

1 Introduction

Database administrators (DBAs) usually need efforts to determine the best index for current applications [16]. It means that finding the best index for the current dataset and workload has always been a challenging task. To some extent, DBAs rely on so-called design consulting tools [2,7,10], which provide advice based on statistical data. However, due to the uncertainty and one sidedness of statistical data [9], DBAs usually find it difficult to make the best choice according to the experiences. With the development of informationization, the size of database increases explosively. As a result, DBAs need a lot of statistical data and efforts to index large-scale data. In order to solve these problems, scholars began to study automatic index recommendation (IR).

Nowadays, we are faced with two major challenges in IR. On the one hand, is that IR consumes a large amount of resources on the massive data, and thus it is impractical to solve IR problem in large scale data by using iterative method [3,5,14]. On the other hand, it is difficult to define the criteria of IR, sometimes IR lower the performance when the criteria selected are inappropriate.

https://doi.org/10.1007/978-981-15-7981-3_1

For example, only taking into account the workload without considering the original data, it is incomplete and sometimes lowers the performance.

IR has always been a challenge problem, which has been studied since 1970, such as literature [15,18] For the first time, reference [3] proposed the IR of RDBMS, which used the concept of "atomic configuration" mentioned in reference [13]. The system iterated from a single index to find all the best index configuration under the whole load. In reference [5,14], the candidate indexes were selected first, and then the indexes were iterated under these candidates. Through the above method, DBAs can find the best index by iterating on the original data set, without statistics and consultation. But it is impossible to find the most suitable index iteratively when there is a large amount of data, since we are unable to afford the cost of iterative recommendation for big data. What's more, the model is not transferable, and it needs to be re-trained when a new dataset is added, which is obviously inefficient.

In recent years, scholars have proposed some solutions based on machine learning [4,16], which can make IR more flexible. Reference [16] proposed a NoDBA system, which used reinforcement learning method for IR. It uses real measurement results or workload prediction results as reward function to determine whether a field should be indexed or not. Reinforcement learning method can automatically recommend index without a large number of data statistics and consultation. In reference [4], the author converted IR to a two-classification problem for improving recommendation precision. Since there is no need for a large number of iterations by using the classification model to determine whether the index is established, the model can greatly reduce the resource consumption. However, the above research still does not solve the problem of model migration. That is, when we want to recommend indexes for datasets, we must rely on the current datasets to train the model. E.g. We need to deal with IR on 10,000,000 datasets and 50,000,000 workloads. Using the model in [4–6,9,13–16] requires training at least 10,000,000 times to get the recommended index of all datasets. In the background of big data, each training will consume a lot of resources, so the mobility of the model has become an important problem to be solved.

To solve the above problems, this paper studies the transferability of the IR model for the first time and proposes a general multi-classification model (CNNIR) to solve the problem. The author considers the influence of dataset features (DF) and workload features (WF) which fundamentally affect IR results. Because the data in real life can always be divided into several kinds of distribution, we build CNNIR on the basis of DF distribution and WF distribution so that CNNIR can achieve better transferability. In summary, this paper firstly studies how to break the gap between different data sets from the perspective of data distribution, and really considers using one IR model to fit all data distribution.

CNNIR is a multi-classification model based on multi-core convolutional neural network. The input of CNNIR is data sequence (DS) and workload sequence (WS), and the output is index configuration containing some fine-grained characteristics of index. CNNIR uses convolution neural network to convolute sequence by multiple convolution kernels for learning features efficiently. The output of CNNIR contains a four-dimensional vector to illustrate the index configuration. It is a multi-classification model, more fine-grained IR, greatly improving the efficiency of the workload.

In summary, this paper makes the following contributions: (1) We propose a new idea from the perspective of data distribution to solve the problem of IR. (2) We propose a technique for characterizing data and workload into vectors. (3) We propose a transferable multi-classification model to solve IR problem. (4) We tested the model's portability on the classic NoSQL MongoDB, and evaluated the accuracy and robustness of CNNIR.

Our paper organized as follows. Section 2 defines the problem and introduces data vectorization. Then we illustrate CNNIR in details in Sect. 3. In Sect. 4, extensive experiments are conducted to test the accuracy and transferability of CNNIR. Finally, we make a conclusion for our contributions in Sect. 5.

2 Featurization

2.1 Problem Definition

With the development of network interconnection, the storage of semi-structured data becomes more and more important, and the IR approaches on semi-structured data could be applied on structured data. In this paper, we consider the IR problem on the semi-structured dataset. The input of this problem is document data (D) and workload (W). The output is the proper index configurations.

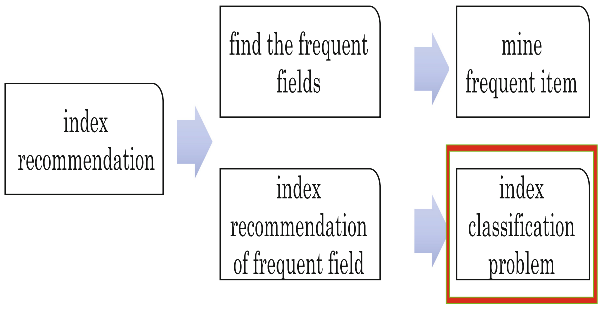
index recommendation
find the frequent fields
mine frequent item
index recommendation of frequent field
index classification problem

Fig. 1. Structure of IC problem

In this paper, the IR problem (see Fig. 1) is transformed into two subproblems. One is frequent field mining. The other is fine-grained index classification (IC) on frequent fields in the red box of Fig. 1. Since W in real life is usually huge, it will inevitably contain many fields with few queries. It is meaningless working on these fields. It can greatly speed up the recommendation by filtering out the fields with few queries in advance. Since the frequent itemset mining has been well studied, we attempt to use the classic frequent itemset

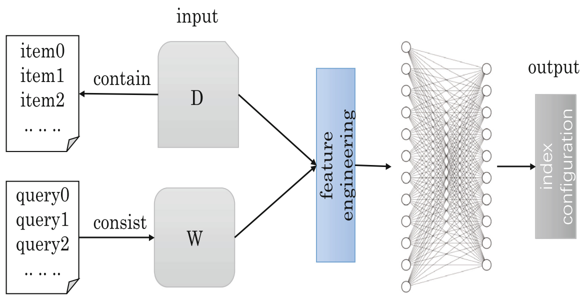

Fig. 2. Structure of IC problem

mining algorithm, FP growth [6] to discover the fields, which accomplish the mining by accessing data twice compared to Apriori [1].

Thus, we focus on the IC problem, as is demonstrated in Fig. 2 in this paper. The input of this problem is D which contains all values of a field, and W which consisted of queries. Then, in the next subsection, we will illustrate the vectorization for normalizing the input of classification task in the red box of Fig. 1.

2.2 Vectorization

We first describe how to vectorized D. Due to the large scale of the D, which is difficult to be directly used for model training, we consider using sampling method. Since stratified sampling can largely retain the features of D, and is widely used in various fields [11,12,19]. In this paper, we use the stratified sampling in the original data as the first step of the vectorization for reducing data size. Our stratified sampling (as shown in the Fig. 3) is divided into two steps: (1) sampling is conducted according to whether the data is empty, (2) sampling is conducted on non-empty data in lexicographic order. The two-level stratified sampling can retain the features of the original data greatly, which is the basis of the best IR.

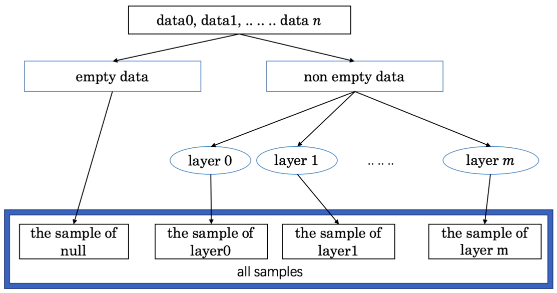

Fig. 3. Structure of stratified sampling

W consists of a series of queries. For the current classification, only the data related to the field affect the index configuration. Therefore, we extract all the data in the workload involved in the current classification as part of the model input. For example, $query0 = \{$name: "Amy"$\}$, $query1 = \{$name: "Tom"$\}$, $query2 = \{$age: 46$\}$. The result of extracting "name" field is ["Amy","Tom"], and the result of extracting "age" field is [46].

For the result of preprocessing W and D, we employ a more fine-grained character (e.g. a to z, 1 to 9) level encoding method [20] as shown in the Fig. 4 to encoding them. As we know, one-hot coding is a common coding method in natural language processing. String-level encoding is very space-consuming. For example, to encode 1000 words, we need one-hot vector of 1000 dimensions in string-level encoding. In big data, data dimension is very high. Therefore, we consider more fine-grained encoding based on characters. The characters in the dictionary are limited, which greatly reduces the sparsity of encoding compared with the string-level one-hot coding.

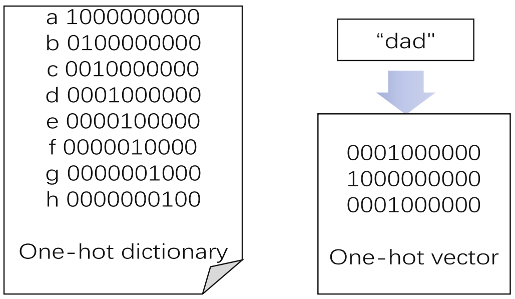

Fig. 4. Sample of coding

3 Classification Model

In this section, we introduce a classification model CNNIR to solve the IC problem defined in Sect. 2.1.

3.1 Classification Task

We transform IC problem in Sect. 2.1 to a classification task. The input of this task is the vector obtained in Sect. 2.2. The output is a four-dimensional vector which illustrates the index configuration of the field. The first dimension indicates whether the field is indexed, the second dimension indicates whether the index is sparse, the third dimension indicates whether the index is partial, and the fourth dimension indicates whether the hash or b-tree index is established. With the above four dimensions as output, we transform IC problem to a classification task, and then build a multi-classification model, as shown in Fig. 5, to solve this problem.

Existing IR method does not consider the transferability of the model. Whenever IR is needed for a new data set, these models always need to be trained. Our model considers from the perspective of data distribution, and studies whether the data with the same distribution will have the same index recommendation. And then, we introduce the CNNIR to achieve above targets. Our model learns the features of data distribution in order to achieve transferability.

For our CNNIR, when it is trained according to D and W under a certain distribution, any data with the same distribution can use this model to processing IR problem. For example, consider two data sets with four workloads, $D = [d_1, d_2]$, $W_{d1} = [w_1, w_2, w_3, w_4]$ and $W_{d2} = [w_1, w_2, w_3, w_4]$ with the same distribution. The index classification model can classify the remaining untrained but equally distributed data (d_2 and W_{d2}) after we use the d_1 and W_{d1} for training.

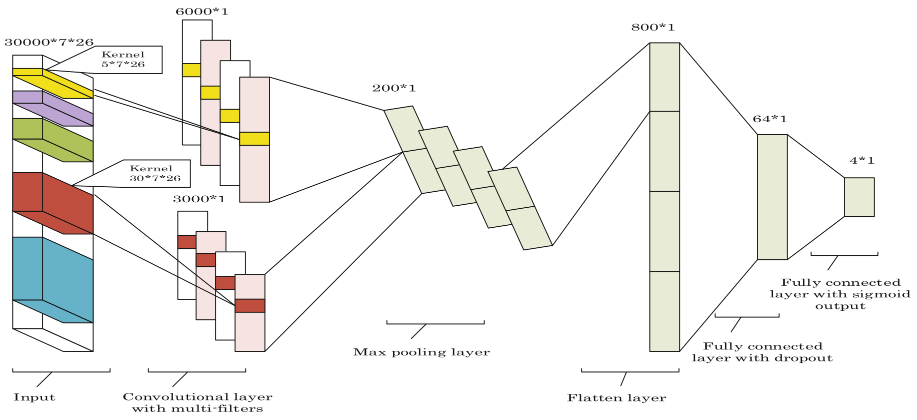
30000*7*26
Kernel 5*7*26
Kernel 30*7*26
6000*1
3000*1
200*1
800*1
64*1
4*1
Input
Convolutional layer with multi-filters
Max pooling layer
Flatten layer
Fully connected layer with dropout
Fully connected layer with sigmoid output

Fig. 5. Structure of CNNIR

3.2 Model Framework

The framework of our solution for the classification problem is shown in Fig. 5, and the input data is the vectorized data obtained in Sect. 2.2. The convolution layer is composed of several convolution kernels with different sizes, which can convolute the information under different steps to achieve better learning effectiveness. The max pooling layer is dynamic in order to make the convolutional output consistent. The length of pooling window is calculated by the blow formula.

$$pool_l = \frac{convshape0}{z}$$

The $convshape0$ is the length of first dimension in convolution output, and z is the normalization parameter. The flatten layer spreads the pool layer output into a one-dimensional vector to connect the full connection layer.

We list more detailed configuration information of models in Table 1. There are five convolution kernels, each of which can be convoluted in a different range and can learn more knowledge representation. Considering the amount of network parameters and representation effectiveness, it is inappropriate that our normalization parameter is too big or too small. Finally, we set $z = 100$ as normalization configuration. As we can see in Table 1, the convolution kernel $[5, 7, 26]$ corresponds to pooling kernel $[60, 1, 1]$ and the convolution kernel $[10, 7, 26]$ corresponds to pooling kernel $[30, 1, 1]$. Then, we can get the same output dimension ([None, 100, 2]) in different convolutional kernel. Additionally, due to the lack of training data, we use dropout [17] technology to avoid over fitting of neural network.

Table 1. Configuration of CNNIR framework

Layers	Configuration	Output shape
Input layer		[None, 30000, 7, 26]
Convolution layer	$[5, 7, 26] * 2$, $[10, 7, 26] * 2$, $[20, 7, 26] * 2$, $[30, 7, 26] * 2$, $[50, 7, 26] * 2$	$[\text{None}, 6000, 1, 1, 2]$, $[\text{None}, 3000, 1, 1, 2]$, $[\text{None}, 1500, 1, 1, 2]$, $[\text{None}, 1000, 1, 1, 2]$, $[\text{None}, 600, 1, 1, 2]$
Max-pooling layer	$[60, 1, 1]$, $[30, 1, 1]$ $[15, 1, 1]$, $[10, 1, 1]$ $[6, 1, 1]$	$[\text{None}, 100, 2]$, $[\text{None}, 100, 2]$ $[\text{None}100, 2]$, $[\text{None}, 100, 2]$ $[\text{None}, 100, 2]$
Flatten layer		$[\text{None}, 1000]$
Full connected layer	64, dropout = 0.1, relu	[None, 64]
Output layer	4, dropout = 0.1, sigmoid	[None, 4]

CNNIR with different kernels improve the accuracy of classification and has a good transferability. Convolutional neural network has been used in image classification field, which can improve the classification accuracy greatly. In this paper, multi-core convolution neural network is used to learn and represent the data and workload. By learning the features of data segments with different convolution steps, we can learn more data distribution features. The index configuration is determined by the original data distribution and workload data distribution of the field and the correlation between the two data sequences. For example, for a query on documents, building a sparse index on a sparse field is better than building a general index. When the query in a workload only involves a part of data, it is better to build a partial index on this field than to build a general index. CNNIR is able to distinguish the above situations by classification method.

3.3 Key Technical Points

In this section, we introduce the key technical points of CNNIR.

(1) Our model has transferability. By using multi-core convolution to learn data distribution features, we can achieve good representation learning for model transferability.

(2) We use character-level one-hot coding instead of string-level representation, which greatly reduces the dimension of input vector.
(3) We define more fine-grained classification output, which can make more detailed recommendation on the index, and greatly improve the efficiency of the workload execution.
(4) Our model has strong expansibility, and we can realize more diverse index classification by redefining the output vector.

4 Experimental Results

Our experiment is divided into two parts, one is to test the accuracy, robustness and convergence of CNNIR (in Sect. 4.2), and the other is to test the IR effectiveness and universality of the CNNIR (in Sect. 4.4). Our experiments run on machines with Intel Core i5 4210U, 8G of RAM and a 480G of disk.

4.1 Datasets

In this paper, we develop a benchmark to generate random document data with various configurations. We use it to generate various data distribution for neural network training and testing. The configurations are as follows: the number of documents, the data distribution (normal distribution or uniform distribution), the number of document fields, the sparsity of each field in the document (if the sparsity is 0.9, only 90% of documents have this field), the cardinality of each field (if the cardinality is 0.1, it means that the cardinality of this field is 0.1 by the number of fields in all documents), and the random seed (used to determine the seed generated by the random number and ensure the repeatability of the experiment). We generate dataset1 and dataset2 which is equally distributed to conduct our experiments.

4.2 Results of Model Performance

In this section we test the accuracy, robustness and convergence of the model. We divide dataset 1 into 80% training set and 20% test set. According to the training set and test set, we test the curve of model loss and the accuracy with the number of iterations under different learning rates (LR). CNNIR has a powerful classification ability through multi-core convolution learning. As shown in the Fig. 9, after 50 iterations, the classification accuracy of the model in the test set is over 95%. As shown in Fig. 6, 7, 8, 9, CNNIR has good robustness and can perform well under various learning rates. We use Adam Optimizer [8] for efficient stochastic optimization to train our model, which only requires first-order gradients with little memory requirement. Figure 9 clearly shows that the model can converge rapidly through several iterations under the learning rate of 0.01, 0.001 and 0.0001.

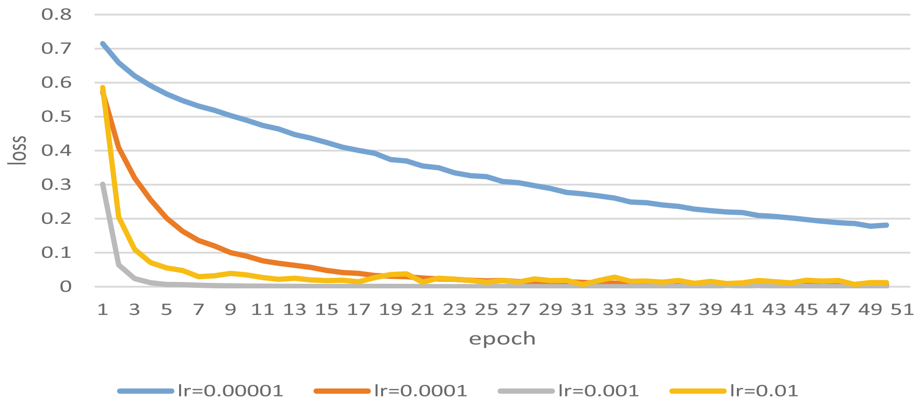

Fig. 6. Loss of different LR in training set

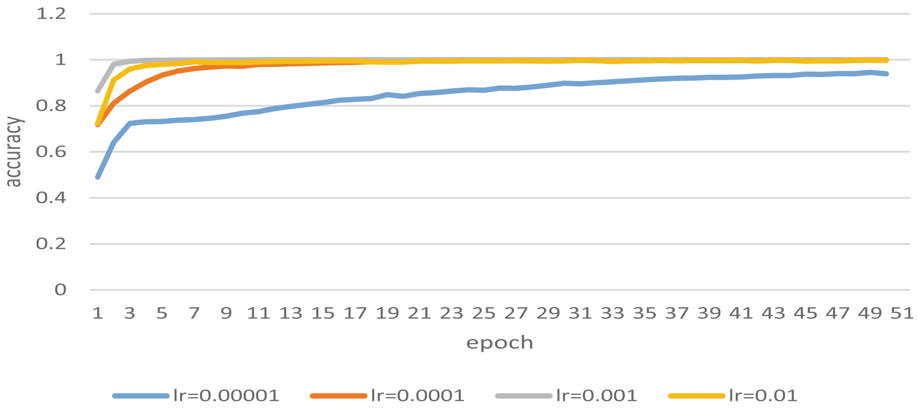

Fig. 7. Accuracy of different LR in training set

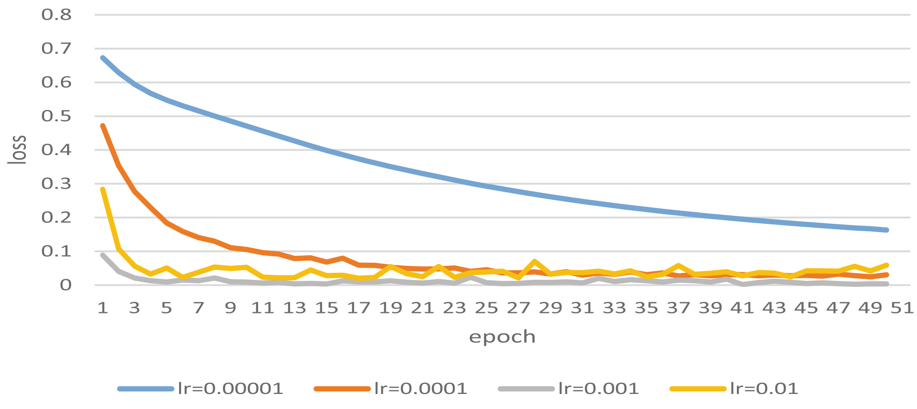

Fig. 8. Loss of different LR in test set

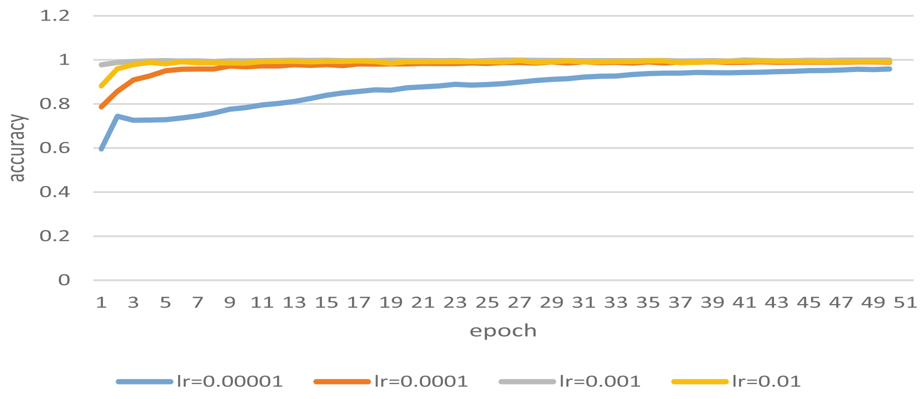

Fig. 9. Accuracy of different LR in test set

4.3 Workload

We use four types of workloads in Table 2 to test the IR efficiency of the model. Theoretically, two conditions should be taken into account when building indexes for these four types of workloads. The first is that when the query only involves a part of data, it is more proper to use a partial index instead of normal index. The second is that it is better to build a B-tree when the query data contains a large number of range queries, and it is better to build a hash index when the query data contains a large number of equivalent queries. We will discuss the IR performance of the model for these workloads in next section.

Table 2. Workloads

Workloads	Selections
W1	10,000 equivalent queries which are evenly distributed in the original data
W2	10,000 range queries which are evenly distributed in the original data
W3	10,000 equivalent queries which are distributed in part of the original data
W4	10,000 range queries which are distributed in part of the original data

4.4 The Experimental Results of Efficiency

In this section, we tested the efficiency on the workloads defined in Sect. 4.3. We use dataset1 to train the model, and then test it with dataset1 and dataset2. Since it takes too long to test all queries in each workload, we randomly select five queries from each workload. As shown in Table 3, through the index configurations from CNNIR, the average execution time of each query can be over 200 times faster than before. What's more, CNNIR only trained on dataset1 also performs well on dataset2, so it has been proved that CNNIR is a general model. That is to say, we only need to train once on CNNIR to make the model suitable for all the data with the same distribution, which can greatly reduce the resource consuming.

Table 3. Workload execution time

Dataset1			Dataset2		
Workloads	NoIndex [ms]	CNNIR [ms]	Workloads	NoIndex [ms]	CNNIR [ms]
W1:Q1	581.43	2.17	W1:Q1	798.80	2.45
W1:Q2	539.74	2.30	W1:Q2	541.10	2.32
W1:Q3	562.95	2.20	W1:Q3	542.76	2.27
W1:Q4	563.45	2.30	W1:Q4	574.98	2.27
W1:Q5	540.77	2.23	W1:Q5	550.04	2.16
W2:Q1	802.64	4.90	W2:Q1	724.54	3.43
W2:Q2	678.20	3.84	W2:Q2	716.98	5.56
W2:Q3	671.12	3.96	W2:Q3	713.30	3.80
W2:Q4	729.89	3.83	W2:Q4	770.02	3.86
W2:Q5	679.75	3.84	W2:Q5	690.09	3.51
W3:Q1	608.23	2.20	W3:Q1	567.71	2.34
W3:Q2	594.85	2.33	W3:Q2	561.83	2.91
W3:Q3	604.54	2.29	W3:Q3	563.07	2.39
W3:Q4	568.63	2.31	W3:Q4	542.59	3.48
W3:Q5	571.95	2.64	W3:Q5	528.28	2.55
W4:Q1	623.17	3.77	W4:Q1	638.14	2.85
W4:Q2	670.29	2.94	W4:Q2	609.24	3.42
W4:Q3	607.25	2.54	W4:Q3	611.63	3.00
W4:Q4	923.00	2.72	W4:Q4	614.00	3.36
W4:Q5	946.13	2.91	W4:Q5	596.52	2.45
Total	653.40	2.91	Total	622.78	3.02

5 Conclusion

This paper focuses on the migration of the IR model in big data. We transform the IR problem into a fine-grained multi-classification problem, and propose a model based on convolutional neural network to solve the classification problem. Two data sets and some workloads for model testing are generated by our benchmark. CNNIR uses the strong representation learning ability of convolutional neural network. As we can see in Sect. 4.2, CNNIR with a strong classification ability achieves over 95% accuracy on the test set. CNNIR is robust and performs well at several learning rates. According to the experimental results in Sect. 4.4, it is demonstrated that CNNIR has excellent universality. CNNIR only trained by dataset1 can achieve high efficiency on untrained dataset2 which has the same distributed dataset with dataset1. In summary, our migration model can reduce the resource consumption of IR by a big margin in big data. In the future, we will consider the case of multi-field IR problem by using Bayesian network to analyze the relationship between each field.

Acknowledgement. This paper was partially supported by NSFC grant U1866602, 61602129, 61772157.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 207–216 (1993)
2. toolpark.de Alle Rechte vorbehalten: The website of powerdesigner (2016). http://powerdesigner.de
3. Chaudhuri, S., Narasayya, V.R.: An efficient, cost-driven index selection tool for Microsoft SQL server. In: VLDB, vol. 97, pp. 146–155. Citeseer (1997)
4. Ding, B., Das, S., Marcus, R., Wu, W., Chaudhuri, S., Narasayya, V.R.: AI meets AI: leveraging query executions to improve index recommendations. In: Proceedings of the 2019 International Conference on Management of Data, pp. 1241–1258 (2019)
5. Hammer, M., Chan, A.: Index selection in a self-adaptive data base management system. In: Proceedings of the 1976 ACM SIGMOD International Conference on Management of Data, Washington, D.C., 2–4 June 1976 (1976)
6. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min. Knowl. Disc. **8**(1), 53–87 (2004). https://doi.org/10.1023/B:DAMI.0000005258.31418.83
7. IDERA, I.L.P.S.G.: The website of er/studio (2004–2020). https://www.idera.com
8. Kinga, D., Adam, J.B.: A method for stochastic optimization. In: International Conference on Learning Representations (ICLR) (2015)
9. Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., Neumann, T.: How good are query optimizers, really? Proc. VLDB Endow. **9**(3), 204–215 (2015)
10. Sparx Systems Pty Ltd.: The website of sparx enterprise architect (2000–2020). https://sparxsystems.com
11. Peng, J., Zhang, D., Wang, J., Jian, P.: AQP++: connecting approximate query processing with aggregate precomputation for interactive analytics. In: Proceedings of the 2018 International Conference on Management of Data (2018)
12. Quoc, D.L., et al.: ApproxJoin: approximate distributed joins. In: Proceedings of the ACM Symposium on Cloud Computing, pp. 426–438 (2018)
13. Finkelstein, S., Schkolnick, M., Tiberio, P.: Physical database design for relational databases. ACM Trans. Database Syst. (TODS) **13**(1), 91–128 (1988)
14. Sattler, K.U., Geist, I., Schallehn, E.: Quiet: continuous query-driven index tuning. In: Proceedings 2003 VLDB Conference, pp. 1129–1132. Elsevier (2003)

15. Schkolnick, M.: The optimal selection of secondary indices for files. Inf. Syst. **1**(4), 141–146 (1975)
16. Sharma, A., Schuhknecht, F.M., Dittrich, J.: The case for automatic database administration using deep reinforcement learning. arXiv preprint arXiv:1801.05643 (2018)
17. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
18. Stonebraker, M.: The choice of partial inversions and combined indices. Int. J. Comput. Inf. Sci. **3**(2), 167–188 (1974). https://doi.org/10.1007/BF00976642
19. Zhang, S., Yao, L., Sun, A., Tay, Y.: Deep learning based recommender system: a survey and new perspectives. ACM Comput. Surv. (CSUR) **52**(1), 1–38 (2019)
20. Zhang, X., Zhao, J., Lecun, Y.: Character-level convolutional networks for text classification (2015)