Supplementary Notes to KARL Library

Tsz Nam Chan   Man Lung Yiu
Department of Computing
Hong Kong Polytechnic University
{cstnchan,csmlyiu}@comp.polyu.edu.hk

Leong Hou U
Department of Computing and Information Science
University of Macau
ryanlhu@umac.mo

I. SOTA Algorithm

SOTA [6] adopts the multi-step approach [8], [1], [2] for evaluating the bound functions (LB and UB), combining with existing indexing structures to support kernel density classification (query type I-τ). We extend their algorithm to support different other types of machine learning models, including approximate kernel density estimation (query type I-ϵ), 1-class SVM (query type II-τ) and 2-class SVM (query type III-τ). We name this algorithm as Multi-Step Kernel Prediction (MSKP).

Algorithm 1 Multi-Step Kernel Prediction (MSKP)

1: **procedure** MSKP(query $\mathbf{q}$, weights $\{w_1, ..., w_n\}$, tree T, threshold τ)
2: Create a max-heap H
3: $e \leftarrow T.R_{root}$
4: $\widehat{lb} \leftarrow LB(\mathbf{q}, e)$, $\widehat{ub} \leftarrow UB(\mathbf{q}, e)$
5: enheap e to H
6: **while** $H \neq \emptyset$ **do**
7: **if** $\widehat{lb} \geq \tau$ **then**
8: return 1
9: **if** $\widehat{ub} < \tau$ **then**
10: return -1
11: $R \leftarrow$ deheap an entry in H
12: $\widehat{lb} \leftarrow \widehat{lb} - LB(\mathbf{q}, e.R)$, $\widehat{ub} \leftarrow \widehat{ub} - UB(\mathbf{q}, e.R)$
13: **if** e is leaf **then**
14: $temp \leftarrow \sum_{\mathbf{p_i} \in e.R} w_i \, \mathcal{K}(\mathbf{q}, \mathbf{p_i})$
15: $\widehat{lb} \leftarrow \widehat{lb} + temp$, $\widehat{ub} \leftarrow \widehat{ub} + temp$
16: **else**
17: **for** each child e_c in e **do**
18: $\widehat{lb} \leftarrow \widehat{lb} + LB(\mathbf{q}, e_c.R)$
19: $\widehat{ub} \leftarrow \widehat{ub} + UB(\mathbf{q}, e_c.R)$
20: enheap e_c to H

Algorithm 1 is only used for the classification-based queries (types I-τ, II-τ and III-τ). For query type I-ϵ, the input threshold τ should be replaced by relative error ϵ. We also need to replace lines 7 to 10 by the following termination condition.

Algorithm 2 Terminiation condition for query type I-ϵ

if $\widehat{ub} \leq (1+\epsilon)\widehat{lb}$ **then**
return $\frac{\widehat{lb}+\widehat{ub}}{2}$

There is another advanced implementation for the termination condition (cf. Algorithm 3), which is based on the unpublished paper of our work [3] in Earth Mover's Distance. This paper is now under submission to TKDE.

Algorithm 3 Advanced terminiation condition for query type I-ϵ [3]

if $\frac{\widehat{ub}-\widehat{lb}}{\widehat{ub}+\widehat{lb}} \leq \epsilon$ **then**
$R = \frac{2\times\widehat{lb}\times\widehat{ub}}{\widehat{lb}+\widehat{ub}}$
return R

II. Relationship between SOTA and KARL

Both SOTA and KARL utilize the same algorithm MSKP. However, compared with SOTA, KARL utilizes tighter bound functions to boost up the efficiency performance. During the implementation of KARL, we only replace LB and UB by our bounding functions [4].

III. Auto-tuning (Offlline)

In Section III-C of our paper [4], we develop the auto-tuning method for obtaining the best index construction in the offline stage. First, we sample 1000 queries from the query dataset as the workload $\mathcal{WL}$. To ensure the fairness, these queries will not be used in the online phase. Then, our algorithm Auto chooses the index from either kd-tree [7] or ball-tree [9] and the most suitable leaf node capacity from the capacity list $CL = \{10, 20, 40, 80, 160, 320, 640\}$ in which the best setting bs provides the fastest running time f_{time} in the selected workload $\mathcal{WL}$.

Algorithm 4 shows the pseudocode of our implementation.

Algorithm 4 Auto-tuning (Offline)

procedure Auto(Workload $\mathcal{WL}$, weights $\{w_1, ..., w_n\}$, tree T, threshold τ, capacity list CL, tree list TL ={kd,ball})
$bs \leftarrow$ null
$f_{time} \leftarrow \infty$
for $t \in TL$ **do**
for $c \in CL$ **do**
$T \leftarrow$ Build tree t with capacity c
$s \leftarrow timer()$
for $q \in \mathcal{WL}$ **do**
$MSKP(\mathbf{q}, weights, T, \tau)$
$e \leftarrow timer()$
$temp \leftarrow e - s$
if $temp \leq f_{time}$ **then**
$bs \leftarrow \{t, c\}$
$f_{time} \leftarrow temp$
Remove tree t
return bs

IV. How to run our codes?

In the file Publish_codes.zip, it contains the codes for all models that we have tested. We have included the shell script for the demonstration of one dataset for each model. In the file Type_I_epsilon, we have included the detailed description of auto-tuning technique (cf. Algorithm 4). The similar script can be also used to handle other types of models.

V. Future direction and Outlook

We have provided the prototype for this KARL library, we will further integrate the codes into one complete library, similar with LibSVM [5]. We expect this library can support more important kernel functions and machine learning models. We hope that this library can help both researchers and practitioners in the industry to experience fast online kernel-based applications (e.g. classification, regression...).

References

[1] T. N. Chan, M. L. Yiu, and K. A. Hua. A progressive approach for similarity search on matrix. In *SSTD*, pages 373–390, 2015.

[2] T. N. Chan, M. L. Yiu, and K. A. Hua. Efficient sub-window nearest neighbor search on matrix. *IEEE Trans. Knowl. Data Eng.*, 29(4):784–797, 2017.

[3] T. N. Chan, M. L. Yiu, and L. H. U. The Power of Bounds: Answering Approximate Earth Movers Distance with Parametric Bounds. *Submitted to IEEE Trans. Knowl. Data Eng.*

[4] T. N. Chan, M. L. Yiu, and L. H. U. KARL: Fast kernel aggregation queries. In *ICDE*, 2019.

[5] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[6] E. Gan and P. Bailis. Scalable kernel density classification via threshold-based pruning. In *ACM SIGMOD*, pages 945–959, 2017.

[7] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann. 2006.

[8] T. Seidl and H. Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD*, pages 154–165, 1998.

[9] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.