Supplementary Notes to KARL

Tsz Nam Chan*†, Man Lung Yiu†, Leong Hou U‡
*Department of Computer Science, The University of Hong Kong
tnchan2@hku.hk
†Department of Computing, Hong Kong Polytechnic University
{cstnchan,csmlyiu}@comp.polyu.edu.hk
‡Department of Computer and Information Science, University of Macau
ryanlhu@umac.mo

I. Introduction

These supplementary notes illustrate some parts which have not been covered in our paper [5] (due to space limitations) but these are essential for readers to know more deeply about KARL. In Section II, we will describe the detailed algorithm for the SOTA method [8]. Then, we illustrate the relationship between SOTA and KARL in Section III. In Section IV, we have the concrete discussion about the auto-tuning method. After that, we discuss how to run our code to reproduce the results in our paper in Section V. Then, we compare the efficiency between KARL and existing approximate 2-class SVM model [1] in the prediction phase in Section VI. Lastly, we provide our future direction and outlook for KARL in Section VII.

II. SOTA Algorithm

SOTA [8] adopts the multi-step approach [13], [2], [3] 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 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.

There is another advanced implementation for the termination condition (cf. Algorithm 3), which is based on the unpublished paper of our work [4] in Earth Mover's Distance. This paper is now under submission to TKDE.

III. 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 [5].

Algorithm 1 Multi-Step Kernel Prediction (MSKP)

```
1: procedure MSKP(query q, weights {w_1, ..., w_n}, tree T, threshold τ)
2:     Create a max-heap H
3:     e ← T.R_root
4:     lb̂ ← LB(q, e), ub̂ ← UB(q, e)
5:     enheap e to H
6:     while H ≠ ∅ do
7:         if lb̂ ≥ τ then
8:             return 1
9:         if ub̂ < τ then
10:            return -1
11:        R ← deheap an entry in H
12:        lb̂ ← lb̂ − LB(q, e.R), ub̂ ← ub̂ − UB(q, e.R)
13:        if e is leaf then
14:            temp ← Σ_{p_i ∈ e.R} w_i K(q, p_i)
15:            lb̂ ← lb̂ + temp, ub̂ ← ub̂ + temp
16:        else
17:            for each child e_c in e do
18:                lb̂ ← lb̂ + LB(q, e_c.R)
19:                ub̂ ← ub̂ + UB(q, e_c.R)
20:                enheap e_c to H
```

Algorithm 2 Terminiation condition for query type I-ϵ

```
1: if ub̂ ≤ (1 + ε) lb̂ then
2:     return (lb̂ + ub̂)/2
```

IV. Auto-tuning (Offlline)

In Section III-C of our paper [5], 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 [12] or ball-tree [14] 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 3 Advanced terminiation condition for query type I-ϵ [4]

```
1: if (ub̂ − lb̂)/(ub̂ + lb̂) ≤ ε then
2:     R = 2 × lb̂ × ub̂ / (lb̂ + ub̂)
3:     return R
```

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

V. 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 (KARL_demo.sh) for the demonstration of one dataset for each model. Unfortunately, the size of each dataset is large and therefore we cannot put them in the github. Please contact the first author to obtain the datasets. In the folder 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.

VI. Additional Experiment: Combination with KARL and Existing Approximate 2-class SVM Models

Many existing literatures [10], [7], [11], [1], [9] in the machine learning community focus on how to boost up the efficiency performance of the prediction phase of 2-class SVM by reducing the number of support vectors (i.e. the data size n). Table I shows the size of different datasets using one support vector reduction method [1] compared with existing software LIBSVM [6]. This technique can significantly reduce the size of datasets, for example: covtype-b (20% of its original dataset size).

TABLE I: Size of datasets (Number of SVs) after training, using Gaussian kernel function

Datasets	LIBSVM	[1]
ijcnn1	9592	2653
a9a	11772	7619
covtype-b	310184	60091

However, most of these support vector reduction techniques, including [1], still remain in $O(nd)$ time complexity in the prediction phase. As such, the throughput can increase by at most 5x, even though the number of SVs is reduced to only 20% of its original dataset size (c.f. Table II). We combine our $\text{KARL}_{\text{auto}}$ in the prediction phase with [1]. The performance can be significantly boosted up which shows that our $\text{KARL}_{\text{auto}}$ can also be combined with approximate SVM techniques with significantly further speed-up.

TABLE II: Throughput (Queries per sec) in the prediction stage

Datasets	LIBSVM	[1]	[1] + $\text{KARL}_{\text{auto}}$
ijcnn1	1170	4192	**1335190**
a9a	610	940	**7391**
covtype-b	38.4	189	**36791**

VII. Future direction and Outlook

In the future, we will further integrate the codes into one complete library, similar with LibSVM [6]. 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] F. Angiulli and A. Astorino. Scaling up support vector machines using nearest neighbor condensation. *IEEE Trans. Neural Networks*, 21(2):351–357, 2010.

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

[3] 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.

[4] T. N. Chan, M. L. Yiu, and L. H. U. The Power of Bounds: Answering Approximate Earth Mover's Distance with Parametric Bounds. *IEEE Trans. Knowl. Data Eng. (To appear)*.

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

[6] 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.

[7] T. Downs, K. E. Gates, and A. Masters. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297, 2001.

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

[9] H. G. Jung and G. Kim. Support vector number reduction: Survey and experimental evaluations. *IEEE Trans. Intelligent Transportation Systems*, 15(2):463–476, 2014.

[10] Y. Lee and O. L. Mangasarian. RSVM: reduced support vector machines. In *SIAM International Conference on Data Mining*, pages 1–17, 2001.

[11] X. Liang. An effective method of pruning support vector machine classifiers. *IEEE Trans. Neural Networks*, 21(1):26–38, 2010.

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

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

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