Journal of Engineering Sciences
Vol 10, Issue 11, Nov /2019
ISSN NO:0377-9254

AN EFFICIENT IMPLEMENTATION OF ONLINE PRODUCT QUANTIZATION

[#1]**P. HEMA VARDHAN**, *M.Tech Student,*
[#2]**SMD SHASHI ULLA**, *Assistant Professor,*
Dept of CSE,
SCIENT INSTITUTE OF TECHNOLOGY, IBRAHIMPATNAM , RANGAREDDY TELANGANA.

Abstract—Approximate nearest neighbor (ANN) search has achieved great success in many tasks. However, existing popular methods for ANN search, such as hashing and quantization methods, are designed for static databases only. They cannot handle well the database with data distribution evolving dynamically, due to the high computational effort for retraining the model based on the new database. In this paper, we address the problem by developing an online product quantization (online PQ) model and incrementally updating the quantization codebook that accommodates to the incoming streaming data. Moreover, to further alleviate the issue of large scale computation for the online PQ update, we design two budget constraints for the model to update partial PQ codebook instead of all. We derive a loss bound which guarantees the performance of our online PQ model. Furthermore, we develop an online PQ model over a sliding window with both data insertion and deletion supported, to reflect the real-time behaviour of the data. The experiments demonstrate that our online PQ model is both time-efficient and effective for ANN search in dynamic large scale databases compared with baseline methods and the idea of partial PQ codebook update further reduces the update cost.

Index Terms:—Online indexing model, product quantization, nearest neighbour search.

I. INTRODUCTION

Computing Euclidean distances between high dimensional vectors is a fundamental requirement in many applications. It is used, in particular, for nearest neighbor (NN) search. Nearest neighbor search is inherently expensive due to the curse of dimensionality [3], [4]. Focusing on the D-dimensional Euclidean space R D, the problem is to find the element NN(x), in a finite set Y ⊂ R D of n vectors, minimizing the distance to the query vector x ∈ R D: NN(x) = arg min y∈Y d(x, y). (1) Several multi-dimensional indexing methods, such as the popular KD-tree [5] or other branch and bound techniques, have been proposed to reduce the search time. However, for high dimensions it turns out [6] that such approaches are not more efficient than the bruteforce exhaustive distance calculation, whose complexity is O(nD). There is a large body of literature [7], [8], [9] on algorithms that overcome this issue by performing approximate nearest neighbor (ANN) search. The key idea This work was partly realized as part of the Quaero Programme, funded by OSEO, French State agency for innovation. It was originally published as a technical report [1] in August 2009. It is also related to the work [2] on source coding for nearest neighbor search. shared by these algorithms is to find the NN with high probability "only", instead of probability 1. Most of the effort has been devoted to the Euclidean distance, though recent generalizations have been proposed for other metrics [10]. In this paper, we consider the Euclidean distance, which is relevant for many applications. In this case, one of the most popular ANN algorithms is the Euclidean Locality-Sensitive Hashing (E2LSH) [7], [11], which provides theoretical guarantees on the search quality with limited assumptions. It has been successfully used for local descriptors [12] and 3D object indexing [13], [11]. However, for real data, LSH is outperformed by heuristic methods, which exploit the distribution of the vectors. These methods include randomized KD-trees [14] and hierarchical k-means [15], both of which are implemented in the FLANN selection algorithm [9]. ANN algorithms are typically compared based on the trade-off between search

quality and efficiency. However, this trade-off does not take into account the memory requirements of the indexing structure. In the case of E2LSH, the memory usage may even be higher than that of the original vectors. Moreover, both E2LSH and FLANN need to perform a final re-ranking step based on exact L2 distances, which requires the indexed vectors to be stored in main memory if access speed is important. This constraint seriously limits the number of vectors that can be handled by these algorithms. Only recently, researchers came up with methods limiting the memory usage. This is a key criterion for problems involving large amounts of data [16], i.e., in large-scale scene recognition [17], where millions to billions of images have to be indexed. In [17], Torralba et al. represent an image by a single global GIST descriptor [18] which is mapped to a short binary code. When no supervision is used, this mapping is learned such that the neighborhood in the embedded space defined by the Hamming distance reflects the neighborhood in the Euclidean space of the original features. The search of the Euclidean nearest neighbors is then approximated by the search of the nearest neighbors in terms of Hamming distances between codes. In [19], spectral hashing (SH) is shown to outperform the binary codes generated by the restricted Boltzmann machine [17], boosting and LSH. Similarly, the Hamming embedding method of Jegou et al. [20], 2 [21] uses a binary signature to refine quantized SIFT or GIST descriptors in a bag-of-features image search framework. In this paper, we construct short codes using quantization. The goal is to estimate distances using vectorto-centroid distances, i.e., the query vector is not quantized; codes are assigned to the database vectors only. This reduces the quantization noise and subsequently improves the search quality. To obtain precise distances, the quantization error must be limited. Therefore, the total number k of centroids should be sufficiently large, e.g., $k = 2^{64}$ for 64-bit codes. This raises several issues on how to learn the codebook and assign a vector. First, the number of samples required to learn the quantizer is huge, i.e., several times k. Second, the complexity of the algorithm itself is prohibitive. Finally, the amount of computer memory available on Earth is not sufficient to store the floating point values representing the centroids. The hierarchical k-means see (HKM) improves the efficiency of the learning stage and of the corresponding assignment procedure [15]. However, the aforementioned limitations still apply, in particular with respect to memory usage and size of the learning set. Another possibility are scalar quantizers, but they offer poor quantization error properties in terms of the trade-off between memory and reconstruction error. Lattice quantizers offer better quantization properties for uniform vector distributions, but this condition is rarely satisfied by real world vectors. In practice, these quantizers perform significantly worse than k-means in indexing tasks [22]. In this paper, we focus on product quantizers. To our knowledge, such a semi-structured quantizer has never been considered in any nearest neighbor search method. The advantages of our method are twofold. First, the number of possible distances is significantly higher than for competing Hamming embedding methods [20], [17], [19], as the Hamming space used in these techniques allows for a few distinct distances only. Second, as a byproduct of the method, we get an estimation of the expected squared distance, which is required for ε-radius search or for using Lowe's distance ratio criterion [23]. The motivation of using the Hamming space in [20], [17], [19] is to compute distances efficiently. Note, however, that one of the fastest ways to compute Hamming distances consists in using table lookups. Our method uses a similar number of table lookups, resulting in comparable efficiency. An exhaustive comparison of the query vector with all codes is prohibitive for very large datasets. We, therefore, introduce a modified inverted file structure to rapidly access the most relevant vectors. A coarse quantizer is used to implement this inverted file structure, where vectors corresponding to a cluster (index) are stored in the associated list. The vectors in the list are represented by short codes, computed by our product quantizer, which is used here to encode the residual vector with respect to the cluster center. The interest of our method is validated on two kinds of vectors, namely local SIFT [23] and global GIST [18] descriptors. A comparison with the state of the art shows that our approach outperforms existing techniques, in particular spectral hashing [19], Hamming embedding [20] and FLANN [9]. Our paper is organized as follows. Section II introduces the notations for quantization as well as the product quantizer used by our method. Section III presents our approach for NN search and Section IV

introduces the structure used to avoid exhaustive search. An evaluation of the parameters of our approach and a comparison with the state of the art is given in Section V.S

II. BACKGROUND: QUANTIZATION, PRODUCT QUANTIZER

A large body of literature is available on vector quantization, see [24] for a survey. In this section, we restrict our presentation to the notations and concepts used in the rest of the paper. A. Vector quantization Quantization is a destructive process which has been extensively studied in information theory [24]. Its purpose is to reduce the cardinality of the representation space, in particular when the input data is real-valued. Formally, a quantizer is a function q mapping a Ddimensional vector x ∈ R D to a vector q(x) ∈ C = {ci ;i ∈ I}, where the index set I is from now on assumed to be finite: I = 0 . . . k − 1. The reproduction values ci are called centroids. The set of reproduction values C is the codebook of size k. The set Vi of vectors mapped to a given index i is referred to as a (Voronoi) cell, and defined as Vi , {x ∈ R D : q(x) = ci}.

The k cells of a quantizer form a partition of R D. By definition, all the vectors lying in the same cell Vi are reconstructed by the same centroid ci . The quality of a quantizer is usually measured by the mean squared error between the input vector x and its reproduction value q(x): MSE(q) = EX d(q(x), x) 2 = Z p(x) d q(x), x2 dx

where d(x, y) = ||x − y|| is the Euclidean distance between x and y, and where p(x) is the probability distribution function corresponding the random variable X. For an arbitrary probability distribution function, Equation 3 is numerically computed using Monte-Carlo sampling, as the average of ||q(x) − x||2 on a large set of samples. In order for the quantizer to be optimal, it has to satisfy two properties known as the Lloyd optimality conditions. First, a vector x must be quantized to its nearest codebook centroid, in terms of the Euclidean distance: q(x) = arg min ci∈C d(x, ci). (4) As a result, the cells are delimited by hyperplanes. The second Lloyd condition is that the reconstruction value must be the expectation of the vectors lying in the Voronoi cell: ci = EX x|i = Z Vi p(x) x dx. (5) The Lloyd quantizer, which corresponds to the kmeans clustering algorithm, finds a near-optimal codebook by iteratively assigning the vectors of a training set to centroids and re-estimating these centroids from the assigned vectors. In the following, we assume that the two Lloyd conditions hold, as we learn the quantizer using k-means. Note, however, that k-means does only find a local optimum in terms of quantization error. Another quantity that will be used in the following is the mean squared distortion ξ(q, ci) obtained when reconstructing a vector of a cell Vi by the corresponding centroid ci . Denoting by pi = P q(x) = ci the probability that a vector is assigned to the centroid ci , it is computed as ξ(q, ci) = 1 pi Z Vi d x, q(x) 2 p(x) dx. (6) Note that the MSE can be obtained from these quantities as MSE(q) = X i∈I pi ξ(q, ci). (7) The memory cost of storing the index value, without any further processing (entropy coding), is ⌈log2 k⌉ bits. Therefore, it is convenient to use a power of two for k, as the code produced by the quantizer is stored in a binary memory. B. Product quantizers Let us consider a 128-dimensional vector, for example the SIFT descriptor [23]. A quantizer producing 64- bits codes, i.e., “only” 0.5 bit per component, contains k = 264 centroids. Therefore, it is impossible to use Lloyd’s algorithm or even HKM, as the number of samples required and the complexity of learning the quantizer are several times k. It is even impossible to store the D × k floating point values representing the k centroids. Product quantization is an efficient solution to address these issues. It is a common technique in source coding, which allows to choose the number of components to be quantized jointly (for instance, groups of 24 components can be quantized using the powerful Leech lattice). The input vector x is split into m distinct sub vectors uj , 1 ≤ j ≤ m of dimension D∗ = D/m, where D is a multiple of m. The sub vectors are quantized separately using m distinct quantizers. A given vector x is therefore mapped as follows: x1, ..., xD∗ | {z } u1(x) , ..., xD−D∗+1, ..., xD | {z } um(x) → q1 u1(x)), ..., qm(um(x) , (8) where qj is a low-complexity quantizer associated with the j th sub vector. With the subquantizer qj we associate the index set Ij , the codebook Cj and the corresponding reproduction values cj,i. A reproduction value of the product quantizer is identified by an element of the product index set I = I1 × . . . × Im. The codebook is therefore defined as the Cartesian product C = C1 × .

. . × Cm, (9) and a centroid of this set is the concatenation of centroids of the m subquantizers. From now on, we assume that all subquantizers have the same finite number k ∗ of reproduction values. In that case, the total number of centroids is given by k = (k ∗) m. (10) Note that in the external case where m = D, the components of a vector x are all quantized separately. Then the product quantizer turns out to be a scalar quantizer, where the quantization function associated with each component may be different. The strength of a product quantizer is to produce a large set of centroids from several small sets of centroids: those associated with the subquantizers. When learning the subquantizers using Lloyd's algorithm, a limited number of vectors is used, but the codebook is, to some extent, still adapted to the data distribution to represent. The complexity of learning the quantizer is m times the complexity of performing k-means clustering with k ∗ centroids of dimension D∗

	memory usage	assignment complexity
k-means	$k\,D$	$k\,D$
HKM	$\frac{b_f}{b_f-1}(k-1)\,D$	$l\,D$
product k-means	$m\,k^*\,D^* = k^{1/m}\,D$	$m\,k^*\,D^* = k^{1/m}\,D$

Table I Memory Usage Of The Codebook And Assignment Complexity For Different Quantizers. Hkm Is Parametrized By Tree Height L And The Branching Factor Bf .

Storing the codebook C explicitly is not efficient. Instead, we store the m × k ∗ centroids of all the subquantizers, i.e., m D∗ k ∗ = k ∗ D floating points values. Quantizing an element requires k ∗D floating point operations. Table I summarizes the resource requirements associated with k-means, HKM and product k-means. The product quantizer is clearly the the only one that can be indexed in memory for large values of k. In order to provide good quantization properties when choosing a constant value of k ∗ , each subvector should have, on average, a comparable energy. One way to ensure this property is to multiply the vector by a random orthogonal matrix prior to quantization. However, for most vector types this is not required and not recommended, as consecutive components are often correlated by construction and are better quantized together with the same subquantizer. As the subspaces are orthogonal, the squared distortion associated with the product quantizer is MSE(q) = X j MSE(qj), (11) where MSE(qj) is the distortion associated with quantizer qj . Figure 1 shows the MSE as a function of the code length for different (m,k ∗) tuples, where the code length is l = m log2 k ∗ , if k ∗ is a power of two. The curves are obtained for a set of 128-dimensional SIFT descriptors, see section V for details. One can observe that for a fixed number of bits, it is better to use a small number of subquantizers with many centroids than having many subquantizers with few bits. At the extreme when m = 1, the product quantizer becomes a regular k-means codebook. High values of k ∗ increase the computational cost of the quantizer, as shown by Table I. They also increase the memory usage of storing the centroids (k ∗ × D floating point values), which further reduces the efficiency if the centroid look-up table does no longer fit in cache memory. In the case where m = 1, we cannot afford using more than 16 bits to keep this cost tractable. Using

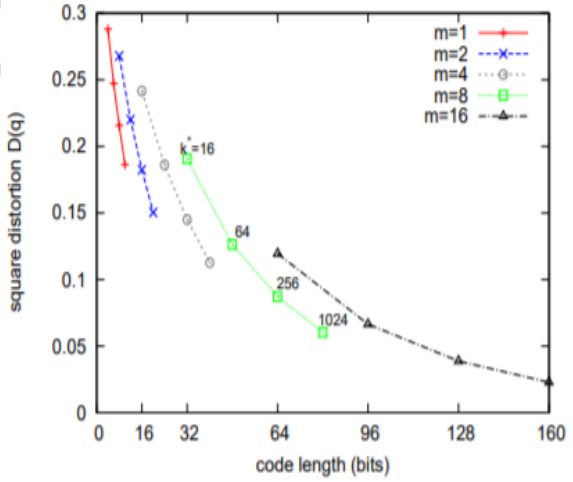

Fig. 1. SIFT: quantization error associated with the parameters m and k ∗ .

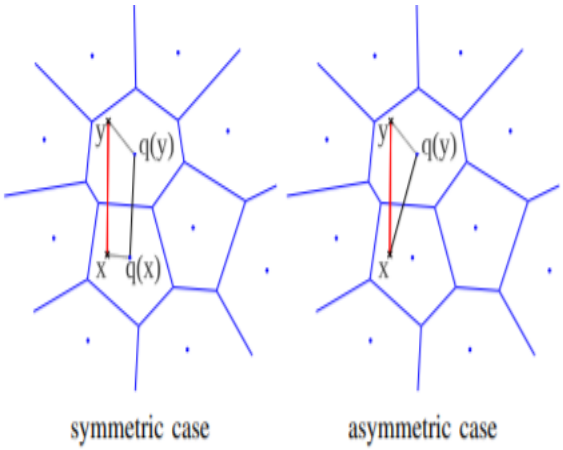

Fig. 2. Illustration of the symmetric and asymmetric distance computation. The distance d(x, y) is estimated with either the distance d(q(x), q(y)) (left) or the distance d(x, q(y)) (right). The mean squared error on the distance is on average bounded by the quantization error.

k ∗ = 256 and m = 8 is often a reasonable choice.

III. PROPOSED SYSTEM

We have presented our online PQ method to accommodate streaming data. In addition, we employ two budget constraints to facilitate partial codebook update to further alleviate the update time cost. A relative loss bound has been derived to guarantee the performance of our model. In addition, we propose an online PQ over sliding window approach, to emphasize on the real-time data. Experimental results show that our method is significantly faster in accommodating the streaming data, outperforms the competing online hashing methods and unsupervised batch mode hashing method in terms of search accuracy and update time cost, and attains comparable search quality with batch mode PQ.

.

IV. ALGORITHM ONLINE PQ

1: initialize PQ with the M ∗ K sub-codewords z 0 1,1 , ..., z0 m,k, ..., z0 M,K using a initial set of data

2: initialize C 0 1,1 , ..., C0 m,k, ..., C0 M,K to be the cluster sets that contain the index of the initial data that belong to the cluster

3: create counters n1,1, ..., nm,k, ..., nM,K for each cluster and initialize each nm,k to be the number of initial data points assigned to the corresponding C 0 m,k

4: for t = 1, 2, 3, ... do

5: get a new data x t

6: partition x t into M subspaces [x t 1 , ..., xt M]

7: in each subspace m ∈ {1, ..., M}, determine and assign the nearest sub-codeword z t m,k for each subvector x t m

8: update the cluster set C t m,k ← C t−1 m,k ∪ {ind} ∀m ∈ {1, ..., M} where ind is the index number of x t

9: update the number of points for each sub-codeword: nm,k ← nm,k + 1 ∀m ∈ {1, ..., M}

10: update the sub-codeword: z t+1 m,k ← z t m,k + 1 nm,k (x t m − z t m,k) ∀m ∈ {1, ..., M}

11: end for

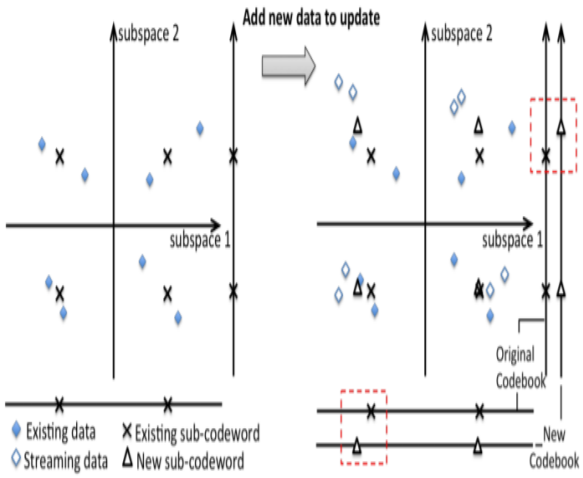

Fig. 3: A schematic figure of online product quantization with budget constraints. There are two subspaces where each subspace has two sub-codewords. After the codebook adapting to the new data, two of the four sub-codewords get hugely changed (highlighted in a red dashed rectangle) and the rest two sub-code words barely changed.

IV. EXPERIMENTS

We conduct a series of experiments on several real-world datasets to evaluate the efficiency and effectiveness of our model. In this section, we first introduce the datasets used in the experiments. We then show the convergence of our online PQ model to the batch PQ method in terms of the quantization error, and then compare the online version and the mini-batch version of our online PQ model. After that, we analyze the impact of the parameters α and λ in update constraints. Finally, we compare our proposed model with existing related hashing methods for different applications.

5.1 Datasets and evaluation criterion

There are one text dataset, four image datasets and two video datasets employed to evaluate the proposed method. 20 Newsgroups Data (News20) [35] consists of chronologically ordered 18,845 newsgroup messages. Caltech-101 [36] consists of 9144 images and each image belongs to one of the 101 categories. Half dome [37] includes 107,732 image patches obtained from Photo Tourism reconstructions from Half Dome (Yosemite). Sun397 [38] contains around 108K images in 397 scenes. Image Net [39] has over 1.2 million images with a total of 1000 classes. YoutubeFaces1 contains 3,425 videos of 1,595 different people, with a total of 621,126 frames. UQ VIDEO2 consists of 169,952 videos with 3,305,525 frames in total. We use 300-D doc2vec features to represent each news article in News20 and 512-D GIST features to represent each image in the four image datasets. We use two different features, 480-D Center-Symmetric LBP (CSLBP) and 560-D Four-Patch LBP (FPLBP) to represent each frame in YoutubeFaces. 162-D HSV feature is used in UQ VIDEO dataset. Table 3 shows detailed statistical information about datasets used in evaluation. We measure the performance of our proposed model by the model update time and the search quality measurement recall@R adopted in [13]. We use recall@20 which indicates that fraction of the query for which the nearest neighbor is in the top 20 retrieved images by the model.

5.2 CONVERGENCE

The data instances in the entire dataset are input sequentially to our online PQ model. We run our algorithm for

1. https://www.cs.tau.ac.il/wolf/ytfaces/
2. http://staff.itee.uq.edu.au/shenht/UQ VIDEO/

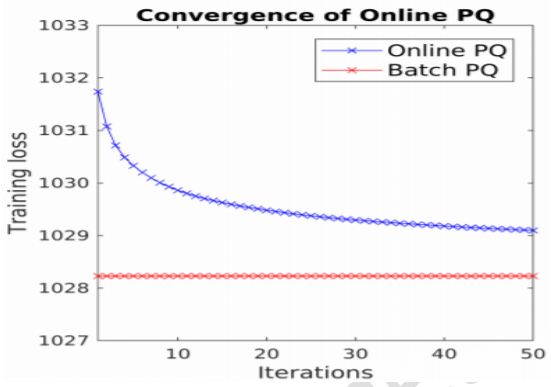

Fig. 5: Convergence of online PQ using ImageNet dataset. Effective iterations are shown on the x-axis.

50 effective iterations3. To show the convergence of our online model, we compare its training loss at each iteration with the one of the batch PQ method. The training loss is computed as the averaged quantization error for all data points in one pass. Figure 5 shows that the training loss of our online model converges to the one of the batch model, implying that codewords learned from the online PQ model are similar to the ones learned from the batch PQ approach. Therefore, the performance of the online PQ model converges to the batch PQ performance.

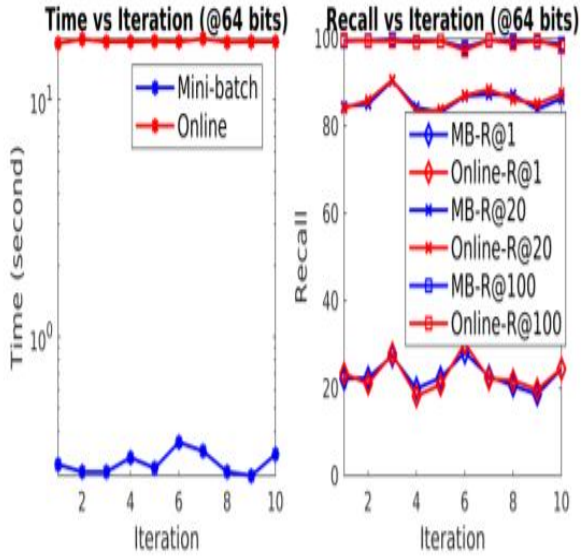

Fig. 6: The left figure shows the update time for each iteration of update. The time of the online version for each iteration sums up the update time of the streaming data corresponding to the ones in the mini-batch.

The right figure shows the recall@1, 20 and 100 for each iteration.

VI.CONCLUSIONS

In this paper, we have presented our online PQ method to accommodate streaming data. In addition, we employ two budget constraints to facilitate partial codebook update to further alleviate the update time cost. A relative loss bound has been derived to guarantee the performance of our model. In addition, we propose an online PQ over sliding window approach, to emphasize on the real-time data. Experimental results show that our method is significantly faster in accommodating the streaming data, outperforms the competing online and batch hashing methods in terms of search accuracy and update time cost, and attains comparable search quality with batch mode PQ. In our future work, we will extend the online update for other MCQ methods, leveraging the advantage of them in a dynamic database environment to enhance the search performance. Each of them has challenges to be effectively extended to handle streaming data. For example, CQ [22] and SQ [23] require the old data for the codewords update at each iteration due to the constant inter-dictionary-elementproduct in the model constraint. AQ [21] requires a high computational encoding procedure, which will dominate the update process in an online fashion. TQ [24] needs to consider the tree graph update together with the codebook and the indices of the stored data. Extensions to these methods can be developed to address the challenges for online update. In addition, online PQ model can be extended to handle other learning problems such as multioutput learning [40], [41]. Moreover, the theoretical bound for the online model will be further investigated.

REFERENCES

1. A. Moffat, J. Zobel, and N. Sharman, “Text compression for dynamic document databases,” TKDE, vol. 9, no. 2, pp. 302–313, 1997.
2. R. Popovici, A. Weiler, and M. Grossniklaus, “On-line clustering for real-time topic detection in social media streaming data,” in SNOW 2014 Data Challenge, 2014, pp. 57–63.
3. A. Dong and B. Bhanu, “Concept learning and transplantation for dynamic image databases,” in ICME, 2003, pp. 765–768.
4. K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, “Online passive-aggressive algorithms,” JMLR, vol. 7, pp. 551–585, 2006.
5. L. Zhang, T. Yang, R. Jin, Y. Xiao, and Z. Zhou, “Online stochastic linear optimization under one-bit feedback,” in ICML, 2016, pp. 392–401.
6. L. Huang, Q. Yang, and W. Zheng, “Online hashing,” in IJCAI, 2013, pp. 1422–1428.
7. “Online hashing,” TNNLS, 2017.
8. M. Ghashami and A. Abdullah, “Binary coding in stream,” CoRR, vol. abs/1503.06271, 2015.
9. C. Leng, J. Wu, J. Cheng, X. Bai, and H. Lu, “Online sketching hashing,” in CVPR, 2015, pp. 2503–2511.
10. F. Cakir and S. Sclaroff, “Adaptive hashing for fast similarity search,” in ICCV, 2015, pp. 1044–1052.
11. Q. Yang, L. Huang, W. Zheng, and Y. Ling, “Smart hashing update for fast response,” in IJCAI, 2013, pp. 1855–1861.
12. F. Cakir, S. A. Bargal, and S. Sclaroff, “Online supervised hashing,” CVIU, 2016.
13. H. Jegou, M. Douze, and C. Schmid, “Product quantization for ´ nearest neighbor search,” TPAMI, vol. 33, no. 1, pp. 117–128, 2011.
14. C. Ma, I. W. Tsang, F. Peng, and C. Liu, “Partial hash update via hamming subspace learning,” IEEE Transactions on Image Processing, vol. 26, no. 4, pp. 1939–1951, 2017.
15. M. Norouzi and D. J. Fleet, “Minimal loss hashing for compact binary codes,” in ICML, 2011, pp. 353–360.
16. W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang, “Supervised hashing with kernels,” in CVPR, 2012, pp. 2074–2081.
17. Y. Gong and S. Lazebnik, “Iterative quantization: A procrustean approach to learning binary codes,” in CVPR, 2011, pp. 817–824.
18. W. Kong and W. Li, “Isotropic hashing,” in NIPS, 2012, pp. 1655– 1663.
19. Y. Weiss, A. Torralba, and R. Fergus, “Spectral hashing,” in NIPS, 2008, pp. 1753–1760.
20. A. Gionis, P. Indyk, and R. Motwani, “Similarity search in high dimensions via hashing,” in VLDB, 1999, pp. 518–529.