Continuously Adaptive Similarity Search

Huayi Zhang*
Worcester Polytechnic Institute
hzhang4@wpi.edu

Lei Cao*
MIT CSAIL
lcao@csail.mit.edu

Yizhou Yan
Worcester Polytechnic Institute
yyan2@wpi.edu

Samuel Madden
MIT CSAIL
madden@csail.mit.edu

Elke A. Rundensteiner
Worcester Polytechnic Institute
rundenst@cs.wpi.edu

ABSTRACT

Similarity search is the basis for many data analytics techniques, including k-nearest neighbor classification and outlier detection. Similarity search over large data sets relies on i) a distance metric learned from input examples and ii) an index to speed up search based on the learned distance metric. In interactive systems, input to guide the learning of the distance metric may be provided over time. As this new input changes the learned distance metric, a naive approach would adopt the costly process of re-indexing all items after each metric change. In this paper, we propose the first solution, called OASIS, to instantaneously adapt the index to conform to a changing distance metric without this prohibitive re-indexing process. To achieve this, we prove that locality-sensitive hashing (LSH) provides an *invariance property*, meaning that an LSH index built on the original distance metric is equally effective at supporting similarity search using an updated distance metric as long as the transform matrix learned for the new distance metric satisfies certain properties. This observation allows OASIS to avoid recomputing the index from scratch in most cases. Further, for the rare cases when an adaption of the LSH index is shown to be necessary, we design an efficient incremental LSH update strategy that re-hashes only a small subset of the items in the index. In addition, we develop an efficient distance metric learning strategy that incrementally learns the new metric as inputs are received. Our experimental study using real world public datasets confirms the effectiveness of OASIS at improving the accuracy of various similarity search-based data analytics tasks by instantaneously adapting the distance metric and its associated index in tandem, while achieving an up to 3 orders of magnitude speedup over the state-of-art techniques.

KEYWORDS

Distance metric learning, Nearest neighbor search, LSH

ACM Reference Format:
Huayi Zhang, Lei Cao, Yizhou Yan, Samuel Madden, and Elke A. Rundensteiner. 2020. Continuously Adaptive Similarity Search. In *2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/mod0251

1 INTRODUCTION

Similarity search is important in many data analytics techniques including i) outlier detection such as distance or density-based models [4, 6, 25, 37]; ii) classification such as k-Nearest Neighbor (kNN) classification [11]; and iii) classical clustering methods such as density-based clustering [16] and micro-clustering [1]. These techniques all rely on similarity search to detect outliers that are far away from their neighbors, to classify testing objects into a class based on the classes of their k-nearest neighbors, or to identify objects that are similar to each other.

The effectiveness and efficiency of similarity search relies on two important yet dependent factors, namely an appropriate *distance metric* and an effective *indexing* method. In similarity search, the distance metric must capture correctly the similarity between objects as per the domain. Pairs of objects measured as similar by the distance metric should indeed be semantically close in the application domain. Indexing ensures the efficiency of the similarity search in retrieving objects similar according to the distance metric. Clearly, the construction of the index relies on the distance metric to determine the similarity among the objects.

*Equal Contribution

SIGMOD '20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/mod0251

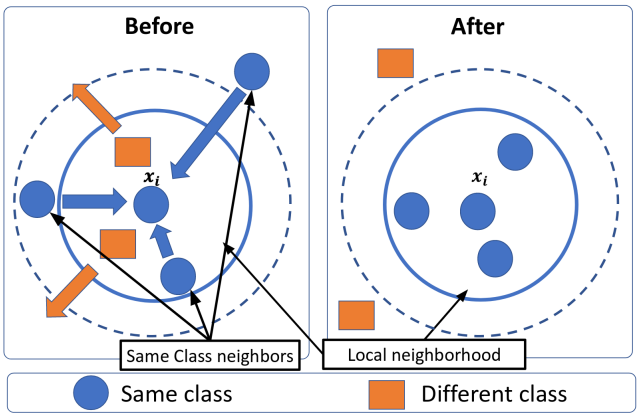


Figure 1: Example of distance metric learning

Unfortunately, general-purpose metrics such as Euclidean distance often do not meet the requirements of applications [19]. To overcome this limitation, distance metric learning [46–49] has been introduced to learn a customized distance metric guided by which pairs of objects the application considers similar or dissimilar, using so called *similarity examples*.

As a motivating example, in a collaboration with a large hospital in the US (name removed due to anonymity), we have been developing a large scale *interactive labelling system* to label EEG segments (450 million segments, 30TB) with 6 classes representing different types of seizures. These labelled EEG segments are used to train a classifier which thereafter can automatically detect seizures based on EEG signals collected during the clinical observation of patients. To reduce the manual labelling effort by the neurologists, our system supports similarity search requests submitted by the neurologists to return the k nearest neighbors of the to-be-labelled segment. The goal is to propagate the labels provided by the experts to similar segments. However, when using Euclidean distance in these nearest neighbor queries, segments may not agree with the returned k nearest neighbors on class labels. To solve this problem, our system employs distance metric learning to construct a custom distance metric from the similarity examples given by the neurologists. This ensures that the nearest neighbors of each segment are more likely to correspond to segments that share the same labels as depicted in Fig. 1. This then empowers us to automatically label EEG segments with higher accuracy.

It has been shown that the effectiveness of distance metric learning depends on the number of similarity examples that are available in the application [5, 19]. Unfortunately, a sufficient number of similarity examples may not be available beforehand to learn an accurate distance metric. Instead, similarity examples may be discovered over time as the users interact with their data. For example, in our above labelling system, whenever the neurologists find a segment that should not be labeled with the same label as its nearest neighbors, the distance metric has to be updated immediately to avoid this mistake in the next search.

As another example, in a financial fraud detection application, analysts may discover new fraudulent transactions not known before. To make sure that the fraud detector can accurately detect the latest threats, its distance metric must be updated according to the new similarity examples, so that the new types of fraud can be separable (distant) from normal objects. Therefore, the distance metric used in similarity search has to be immediately learned and updated as soon as new similarity examples become available.

Challenges & State-of-the-Art. While the continuous learning of a distance metric to be used in similarity search has been studied, it has been overlooked in the literature that a change in distance metric renders the index constructed using the old distance metric ineffective. This is because the objects co-located by the index may no longer be close under the new distance metric. Therefore, the index must be re-built whenever the distance metric changes to assure the users always work with the most up-to-date distance semantics. Note that unlike index updates triggered by the insertion of new objects, a change in the distance metric may influence all existing objects. However, re-building an index from scratch is known to be an expensive process, often taking hours or days [21]. Although this cost may be acceptable in traditional databases where an index is typically built only once upfront, in our context where the distance metric may change repeatedly this is not viable. In particular, in interactive systems such as our labelling system, hours of quiescent time due to rebuilding index often is not acceptable to the neurologists, because the time of the neurologists is precious. However, without the support of an index, similarity search is neither scalable nor efficient. In other words, in similarity search the need to quickly *adapt* to the latest semantics conflicts with the need to be *efficient*.

Further, like indexing, distance metric learning itself is an expensive process [47, 48]. Repeatedly learning a new distance metric from scratch every time when a similarity example becomes available would introduce a tremendous computational overhead especially when new examples arrive frequently. Although some online distance metric learning methods have been proposed [9, 24, 41], these techniques [24] either tend to be not efficient due to the repeated computation of the all-pair distances among all objects involved in all similarity examples received so far, or others [9, 41] directly adopt the update method used in solving the general online learning problem [13]. This can lead to an invalid distance metric producing negative distance values [24].

Our Proposed Approach. In this work. we propose the first continuously adaptive similarity search strategy, or OASIS for short. It uses an incremental distance metric learning

method that together with an efficient index update strategy satisfies the conflicting requirements of adaptivity and efficiency in similarity search.

• OASIS models the problem of incremental distance metric learning as an optimization problem of minimizing a regularized loss [13, 41]. This enables OASIS to learn a new distance metric based on only the current distance metric and the expected distances between the objects involved in the newly provided similarity example. Therefore, it efficiently updates the distance metric with complexity quadratic in the dimensionality of the data – independent of the total number of objects involved in all similarity examples (training objects).

• We observe that the popular Locality Sensitive Hashing (LSH) [17, 23, 31, 32], widely used in online applications to deal with big data, is robust against changes to the distance metric, such that in many cases it is not necessary to update the LSH index even when the distance metric changes. Distance metric learning transforms the original input feature space into a new space using a *transform matrix* learned from the similarity examples. We formally prove in Sec. 5 that hashing the objects in the original input space is equal to hashing the objects in the new space using a family of new hash functions converted from the original hash functions. Therefore, if the new hash functions remain locality-sensitive – i.e., similar objects are mapped to the same "buckets" with high probability, then the original LSH index will continue to remain effective in supporting similarity search also with the new distance metric. We show that this holds as long as the transform matrix learned by distance metric learning satisfies certain statistical properties. Leveraging this *invariance observation*, OASIS avoids the unnecessary recomputation of LSH hashes.

• Further, we design an efficient method to adapt the LSH index when an update is found to be necessary. It is built on our finding that the identification of the objects that most likely would move to different buckets after the distance metric changes can be transformed into the problem of finding the objects that have a *large inner product* with some *query objects*. Therefore, the objects that have to be updated in the index can be quickly discovered by submitting a few queries to a compact inner product-based LSH index. This way, OASIS effectively avoids re-hashing the large majority of objects during each actual LSH update.

Our experimental study using public datasets and large synthetic datasets confirms that OASIS is up to 3 orders of magnitude faster than approaches that use traditional online metric learning methods [24] and update the index whenever the distance metric changes. Moreover, OASIS is shown to significantly improve the accuracy of similarity search-based data analytics tasks from outlier detection, classification, labeling, and nearest neighbor-based information retrieval compared to approaches that do not update the distance metric or use traditional online metric learning methods.

2 PRELIMINARIES

2.1 Distance Metric Learning

The goal of distance metric learning is to learn a distance function customized to the specific semantics of similarity between objects of the application. Typical distance metric learning techniques [46–49] achieve this by learning a Mahalanobis distance metric D_M, defined by:

$$D_M(\overrightarrow{x_i}, \overrightarrow{x_j}) = (\overrightarrow{x_i} - \overrightarrow{x_j})^T M (\overrightarrow{x_i} - \overrightarrow{x_j}) \quad (1)$$

Here $\overrightarrow{x_i}$ and $\overrightarrow{x_j}$ denote two d-dimensional objects, while M represents a $d \times d$ matrix, called the *transform matrix*. The Mahalanobis metric can be viewed as: (1) applying first a *linear transformation* from the d-dimensional input space R_o^d to a new d-dimensional input space R_n^d represented by $L : R_o^d \rightarrow R_n^d$; followed by (2) computing the Euclidean distance in the new input space as in Equation 2:

$$D_M(\overrightarrow{x_i}, \overrightarrow{x_j}) = \left\| L(\overrightarrow{x_i} - \overrightarrow{x_j}) \right\| \quad (2)$$

where $L^T L = M$.

M must be *positive semidefinite*, i.e., have no negative eigenvalues [19], to assure that the distance function will return positive distance values. Distance metric learning learns the transform matrix M based on the user inputs regarding which items are or are not considered to be similar, so called similarity examples. The learned transform matrix aims to ensure that all user-provided *similarity examples* are satisfied in the transformed space.

The particular format of the similarity examples may differ from application to application. In kNN classification [11] and outlier detection [25], users may explicitly tell us which class some examples belong to. KNN classification assigns an object to the class most common among its k nearest neighbors discovered from the labeled training examples. Therefore, to improve the classification accuracy, given an example $\overrightarrow{x_i}$, distance metric learning aims to learn a distance metric that makes $\overrightarrow{x_i}$ to share the same class with its k nearest neighbor examples as shown in Fig. 1. This is also what is leveraged by the neurologists who specify kNN queries in our motivating example about the online labeling system. In distance-based outlier detection [25], outliers are defined as the objects that have fewer than k neighbors within a distance range d. Distance metric learning therefore aims to learn a transform matrix such that in the transformed space the distance between an outlier example and its kth nearest neighbor is larger than d. In clustering, users may tell us that two example objects (or examples) either must be or cannot be in the same cluster.

In the literature, different distance metric learning methods have been proposed for various applications, such as Large Margin Nearest Neighbor (LMNN) [47, 48] for kNN classification, MELODY [51] for outlier detection, and DML [49] for clustering.

2.2 Locality Sensitive Hashing

Similarity search on large data relies on indexing, because many search operations, including k nearest neighbor (kNN) search, are expensive. In online applications or interactive systems which have stringent response time requirements, indexing that produces approximate results tends to be sufficient as long as it comes with accuracy guarantee; in particular, locality sensitive hashing (LSH) [17, 23, 31, 32], is a popular method because it is scalable to big data, effective in high dimensional space, and amenable to dynamic data, etc.

LSH hashes input objects so that similar objects map to the same "buckets" with high probability. The number of buckets is kept much smaller than the universe of possible input items. To achieve this goal, LSH uses a family of hash functions that make sure the objects close to each other have a higher probability to collide than the objects far from each other, so-called *locality-preserving hashing*. Here we introduce a popular type of hash function used in LSH:

$$f(\overrightarrow{x}) = \left\lfloor \frac{\overrightarrow{a} \cdot \overrightarrow{x} + b}{W} \right\rfloor \tag{3}$$

where $\overrightarrow{a}$ is a random vector whose entries are drawn from a normal distribution. This ensures $f(\overrightarrow{x})$ is locality-preserving [12]. b is a random constant from the range $[0, W)$, with W a constant that denotes the number of the buckets.

Next, we describe the process of **building** a classical (K,N)-parameterized LSH index:

• Define an LSH family $\mathcal{F}$ of locality-preserving functions $f: R^d \rightarrow \mathcal{S}$ which map objects from the d-dimensional input space R^d to a bucket $s \in \mathcal{S}$.

• Given the input parameter K and N, define a new LSH family $\mathcal{G}$ containing N meta hash functions $g_n \in \mathcal{G}$, where each meta function g_n is constructed from K random hash functions f_1, f_2, ..., f_k from $\mathcal{F}$. Each g_n corresponds to one hash table.

• Hash objects: given an input object $\overrightarrow{x}$, $g_n(\overrightarrow{x}) = (f_1(\overrightarrow{x}), f_2(\overrightarrow{x}), ..., f_K(\overrightarrow{x}))$ $(1 \leq n \leq N)$.

The N hash tables produced by the N meta hash functions correspond to the index structure of the given input data.

The procedure of **querying** the nearest neighbors of an object $\overrightarrow{q}$ from an LSH index is:

• Given an object $\overrightarrow{q}$, for each hash function $g_n \in \mathcal{G}$, calculate $g_n(q)$, and return the objects $\overrightarrow{x}$, where $g_n(\overrightarrow{x}) = g_n(\overrightarrow{q})$, that is, the objects $\overrightarrow{x}$ that fall into the same bucket with $\overrightarrow{q}$.

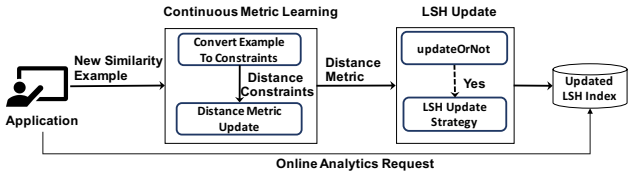


Figure 2: The OASIS System

• Union the results returned from all hash functions g_n (n: 1 ... N) as candidates into one set, and find the nearest neighbors of $\overrightarrow{q}$ from this candidate set.

In addition to kNN queries, LSH is commonly used to speed up data analytics techniques [40] for which similarity search plays a critical role. Examples include distance or density-based outliers [4, 6, 25, 37], density-based or spectral clustering [16, 30]. In these applications, exact results of similarity search that would require significantly more retrieval time are not necessarily better than approximate results with an accuracy guarantee as produced by LSH.

2.3 Problem Statement

We target the scenario where new similarity examples (e.g., user labels) continuously become available, requiring an update to the similarity metric and the associated index.

Definition 2.1. **Continuously Adaptive Similarity Search.** Given the transform matrix M_t used in the current distance metric D_{M_t} and the current LSH index H_t produced by LSH family $\mathcal{F}_t$ w.r.t. distance metric D_{M_t}, after receiving a new similarity example E at time step t, our goal is that:

(1) The transform matrix M_t be updated to M_{t+1} such that the new similarity example E is satisfied by the new distance metric $D_{M_{t+1}}$ using M_{t+1}; and

(2) The LSH index H_t be updated to H_{t+1} so to be equivalent to re-hashing the data using a new LSH family $\mathcal{F}'$ which is locality-preserving w.r.t. the new distance metric $D_{M_{t+1}}$.

3 THE OVERALL OASIS APPROACH

As shown in Fig. 2, given a similarity example supplied by a particular application, the continuous metric learning component of OASIS first converts it into one or multiple distance constraints in a standard format (Sec. 4.1). Then, our incremental learning strategy updates the distance metric to satisfy these constraints (Sec. 4.2). Taking the new distance metric as input, the LSH update component determines if the LSH index has to be updated based on our invariance observation established in Lemma 5.2 (Sec. 5). If necessary, OASIS invokes our two-level LSH-based update strategy (Sec. 6) to update the LSH index according to the new distance metric.

Applications. OASIS can be used to support a broad range of applications, including but not limited to kNN classification [11] used in text categorization and credit rating [19, 22], distance-based outlier detection [7, 8] to detect

network intrusion or financial frauds [15], and information retrieval supported by kNN queries to retrieve documents, images, or products that are most relevant to the given object. LSH indexes are often used in these applications to speed up the neighbor search process as discussed in Sec. 2.2.

In these applications, the new similarity examples might correspond to the objects mis-classified by the kNN classifier, the outliers that are captured recently and believed to be different from the existing outliers, or a retrieved object not relevant to the query object. OASIS then adapts the existing distance metric to avoid these newly observed classification errors, capture the new type of outliers, or ensure that only relevant objects are returned. Next, the LSH index will be updated to effectively support the new queries subsequently submitted by the users or to sequentially process new data, as shown in Fig. 2.

4 CONTINUOUS DISTANCE METRIC LEARNING

In this section, we show how different types of similarity examples in various similarity search-based applications can be encoded into the distance constraints in a quadruple format (Sec. 4.1). In Sec. 4.2, we introduce an incremental distance metric learning strategy using the *distance constraints* as input. This general method serves diverse applications.

4.1 Modeling the Similarity Examples

In OASIS, a similarity example is converted into one or multiple distance constraints in quadruple format $C = (\overrightarrow{u}, \overrightarrow{v}, dt, b)$ that place constraints on the distance between two objects. More specifically, in C, dt corresponds to the target distance between objects $\overrightarrow{u}$ and $\overrightarrow{v}$. When the binary flag b is -1, the distance between examples $\overrightarrow{u}$ and $\overrightarrow{v}$ cannot be larger than dt. When b is 1, the distance between $\overrightarrow{u}$ and $\overrightarrow{v}$ should be larger than dt.

Next, we show how to convert different types of similarity examples in different analytics tasks into distance constraints. To provide some representative examples, we illustrate the conversion principles for kNN classifiers, kNN queries, and distance-based outliers.

4.1.1 Distance Constraints in kNN Classifier

In a kNN classifier, given a new example $\overrightarrow{q}$, the new constraints can be formed based on $\overrightarrow{q}$'s nearest neighbors. The intuition is that the nearest neighbors of $\overrightarrow{q}$ should be the objects that belong to the same class as $\overrightarrow{q}$. This will improve the classification accuracy of the kNN classifier.

First, we define the concept of *target neighbor*.

Definition 4.1. Given an object $\overrightarrow{q}$, its target neighbors denoted as $\mathbb{N}^t(\overrightarrow{q})$ correspond to the k nearest neighbors, determined by the current learned distance metric D_M, that share the same label y_i with $\overrightarrow{q}$ in the labeled example dataset.

The distance constraints are formed based on the nearest neighbor of $\overrightarrow{q}$ and its target neighbors as shown in Def. 4.2.

Definition 4.2. Given the new labeled object $\overrightarrow{q}$ and its nearest neighbor with a class label that is distinct from $\overrightarrow{q}$'s label, denoted as $N^n(\overrightarrow{q})$, $\forall$ target neighbors of $\overrightarrow{q}$ denoted as $N_i^t(\overrightarrow{q}) \in \mathbb{N}^t(\overrightarrow{q})$, a new constraint C_i is formed as: $C_i = (\overrightarrow{q}, N_i^t(\overrightarrow{q}), D_M(\overrightarrow{q}, N^n(\overrightarrow{q})), -1)$ holds, where D_M denotes the distance computed using the current learned distance metric.

In Lemma 4.3, we show that the new distance metric learned from these constraints is able to improve the classification accuracy of the kNN classifier.

Lemma 4.3. *Given an object $\overrightarrow{q}$ previously mis-classified by the kNN classifier, $\overrightarrow{q}$ will no longer be mis-classified if the new distance metric parameterized by transform matrix M_{t+1} satisfies the new constraints $\mathbb{C} = \{C_1, C_2, ..., C_k\}$ defined by Def. 4.2.*

Proof. By the new constraints $\mathbb{C}$, the distance between the new mis-classified object $\overrightarrow{q}$ $(\overrightarrow{u_t})$ and any of its target neighbors $N_i^t(\overrightarrow{q})$ $(\overrightarrow{v_t})$ has to be smaller than the distance between $N_i^t(\overrightarrow{q})$ and its nearest neighbor with non-matching label $N^n(\overrightarrow{q})$ (dt_t), then the target neighbors of $\overrightarrow{q}$ are guaranteed to be the k nearest neighbors of $\overrightarrow{q}$ in the labeled example set dt. By the definition of the kNN classifier, $\overrightarrow{q}$ belongs to the class of its target neighbors which share the same class with $\overrightarrow{q}$. Therefore, now $\overrightarrow{q}$ is guaranteed to be correctly classified. Lemma 4.3 is proven. □

Apply to k Nearest Neighbor (kNN) Queries. The above constraint definition can be equally applied to support kNN queries. Same as in the case of kNN classification, in a kNN query, a small set of labeled examples are collected to learn a distance metric that makes sure an example object $\overrightarrow{q}$ is in the same class with its nearest neighbors found in the example set. The key difference between kNN queries and kNN classification is that in kNN queries the similarity search is conducted on the large input dataset instead of the labeled example set, although its distance metric is learned based on the example set.

4.1.2 Distance Constraints in Distance-based Outliers

As described in Sec. 2.1, in the classical distance-based outlier definition [25], outliers are defined as the objects that have fewer than k neighbors within a distance range d. Similar to kNN classification, OASIS forms the distance constraints based on the k nearest neighbors of the new received example $\overrightarrow{q}$ as shown in Def. 4.4.

Definition 4.4. Given the new labeled object $\overrightarrow{q}$ and its k nearest neighbors denoted as $\mathbb{KNN}(\overrightarrow{q})$, $\forall$ object $\overrightarrow{n_i} \in \mathbb{KNN}(\overrightarrow{q})$, a new constraint $C_i = (\overrightarrow{q}, \overrightarrow{n_i}, d, 1)$ is formed if $\overrightarrow{q}$ is an outlier. Otherwise, $C_i = (\overrightarrow{q}, \overrightarrow{n_i}, d, -1)$.

Using the distance metric learned from the constraints formed by Def. 4.4, the distance-threshold outliers can be effectively detected in the newly transformed space. Intuitively, given an object $\overrightarrow{q}$ which is an inlier, but was mis-classified as outlier by the outlier detector using the old distance metric in the old space, it will no longer be mis-classified in the new space. This is because in the new space the distances between $\overrightarrow{q}$ and its k nearest neighbors are no larger than d. Therefore, $\overrightarrow{q}$ has at least k neighbors now. On the other hand, if $\overrightarrow{q}$ is an outlier, but was missed by the outlier detector in the old space, it tends to be captured in the new metric space. This is because now the distance between $\overrightarrow{q}$ and its k nearest neighbors are larger than d. Therefore, it is unlikely that $\overrightarrow{q}$ has k neighbors.

4.2 Incremental Learning Strategy

The incremental learning strategy in OASIS takes one distance constraints $C_t = (\overrightarrow{u_t}, \overrightarrow{v_t}, dt_t, b_t)$ as input. First, it compute the distance $\hat{dt}_t = D_{M_t}(\overrightarrow{u_t}, \overrightarrow{v_t})$ based on the current distance metric using the transform matrix M_t. It incurs a loss $l(\hat{dt}_t, dt_t^*)$, where $dt_t^* = dt_t(1 + b_t\gamma)$ corresponds to the target distance dt_t weighted by $\gamma \in (0,1)$. If b_t is -1, then the weighted target distance dt_t^* is smaller than the target distance dt_t specified in C. Otherwise, dt_t^* is larger than dt_t. Therefore, $l(\hat{dt}_t, dt_t^*)$ measures the difference between the current distance and the expected distance between the examples specified in C_t.

Then OASIS updates the matrix from M_t to M_{t+1} under the optimization goal of minimizing the total loss of all distance constraints received so far, i.e,

$$Loss = \sum_t l(\hat{dt}_t, dt_t^*) \tag{4}$$

The goal is to ensure that the constraints seen previously are still satisfied. OASIS uses a typical model for the loss function, that is, the squared loss [19]: $l(\hat{dt}_t, dt_t^*) = \frac{1}{2}(\hat{dt}_t - dt_t^*)^2$, though others could be plugged in.

Leveraging the state-of-the-art online learning strategy [13, 41], OASIS solves the problem of minimizing Eq. 4 by learning a M_{t+1} that minimizes a regularized loss:

$$M_{t+1} = \text{argmin} D_{ld}(M_{t+1}, M_t) + \eta l(\bar{dt}, dt_t^*) \tag{5}$$

In Eq. 5, $D_{ld}(M_{t+1}, M_t)$ is a regularization function given by $D_{ld}(M_{t+1}, M_t) = tr(M_{t+1}M_t^{-1})$ - logdet$(M_{t+1}M_t^{-1})$ - d. Here the LogDet divergence [10] is used as the regularization function due to its success in metric learning [13]. $\bar{dt} = \delta_t^T M_{t+1}\delta_t$ corresponds to the distance between $\overrightarrow{u_t}$ and $\overrightarrow{v_t}$ computed using the new transform matrix M_{t+1}, where $\delta_t = \overrightarrow{u_t} - \overrightarrow{v_t}$. $\eta > 0$ denotes the regularization parameter, and d corresponds to the data dimension.

By [38], Eq. 5 can be minimized if:

$$M_{t+1} = M_t - \frac{\eta(\bar{dt} - dt_t^*)M_t\delta_t\delta_t^T M_t}{1 + \eta(\bar{dt} - dt_t^*)\delta_t^T M_t\delta_t} \tag{6}$$

The matrix M_{t+1} resulting from Eq. 6 is guaranteed to be *positive definite* [24]. This satisfies the requirement of distance metric learning.

Next, we show how to compute M_{t+1} using Eq. 6. First, we have to compute $\bar{dt}$. Since $\bar{dt}$ is a function of M_{t+1}, to compute $\bar{dt}$, we have to eliminate M_{t+1} from its computation first. By multiplying Eq. 6 by δ_t^T and δ_t on right and left, we get: $\delta_t^T M_{t+1}\delta_t = \delta_t^T M_t\delta_t - \frac{\eta\delta_t^T(\bar{dt}-dt_t^*)M_t\delta_t\delta_t^T M_t\delta_t}{1+\eta(\bar{dt}-dt_t^*)\delta_t^T M_t\delta_t}$.

Since $\bar{dt} = \delta_t^T M_{t+1}\delta_t$ and $\hat{dt}_t = \delta_t^T M_t\delta_t$, we get:

$$\bar{dt} = \frac{\eta dt_t^*\hat{dt}_t - 1 + \sqrt{(\eta dt_t^*\hat{dt}_t - 1)^2 + 4\eta\hat{dt}_t^2}}{2\eta\hat{dt}_t} \tag{7}$$

Using Eq. 7, we can directly compute $\bar{dt}$. Next, M_{t+1} can be computed by plugging Eq. 7 into Eq. 6.

Algorithm 1 Incremental Distance Metric Learning

```
1: function UpdateMetric(Current Metric M_t, C_t = (u_t, v_t, dt_t, b_t), γ
2:     dt̂_t = D_{M_t}(u_t, v_t);
3:     dt*_t = dt_t(1 + b_t γ);
4:     Compute dt̄ using Eq. 7;
5:     Compute M_{t+1} using Eq. 6;
```

The main steps of the incremental distance metric learning are shown in Alg. 1.

Time Complexity Analysis. The time complexity of this method is determined by the product operation between the d-dimensional vector and $d \times d$ matrix. It is quadratic in the number of dimensions d, but does not rely on the number of distance constraints received so far nor on the number of example objects involved in these constraints.

5 LOCALITY SENSITIVE HASHING: INVARIANCE OBSERVATION

In this section, we will establish the *invariance observation* for LSH, stating that in many cases it is not necessary to update the LSH index at all when the distance metric changes. That is, we will show that an LSH index built on the Euclidean distance is sufficient to support similarity search under the learned distance metric as long as the eigenvalues of its transform matrix M fall into a certain range.

For this, we first show that the original LSH index H built using a LSH family $\mathcal{F}$ can be considered as a LSH index H' with respect to the learned distance metric D_M built using a different family of hash functions $\mathcal{F}'$.

Lemma 5.1. *Given an LSH index H w.r.t. Euclidean distance built using a family of locality-preserving hash functions:*

$$f(\overrightarrow{u}) = \left\lfloor \frac{\overrightarrow{a}\,\overrightarrow{u} + b}{W} \right\rfloor \tag{8}$$

then H can be considered to be an LSH index w.r.t the learned distance metric D_M with transform matrix $M = L^T L$, built using the following family of hash functions:

$$f'(\overrightarrow{u'}) = \left\lfloor \frac{\overrightarrow{a}'\overrightarrow{u'} + b}{W} \right\rfloor \tag{9}$$

where $\overrightarrow{a}' = \overrightarrow{a}L^{-1}$ and $\overrightarrow{u'} = L\overrightarrow{u}$. [1]

Proof. Given a transform matrix M in the learned distance metric, we get a L, where $M = L^T L$. Replacing $\overrightarrow{a}$ with $\overrightarrow{a}L^{-1}L$ ($\overrightarrow{a} = \overrightarrow{a}L^{-1}L$) from Eq. 8, we get: $f(\overrightarrow{u}) = \left\lfloor \frac{\overrightarrow{a}L^{-1}L\overrightarrow{u}+b}{W} \right\rfloor$.

Considering $\overrightarrow{a}L^{-1}$ as a new hyperplane $\overrightarrow{a}'$, we get $f(\overrightarrow{u}) = \left\lfloor \frac{\overrightarrow{a}'L\overrightarrow{u}+b}{W} \right\rfloor$. By the definition of the Mahalanobis distance (Sec. 2), $L\overrightarrow{u}$ corresponds to the object in the new space transformed from $\overrightarrow{u}$. Replacing $L\overrightarrow{u}$ with $\overrightarrow{u}'$, we get $f'(\overrightarrow{u}') = \left\lfloor \frac{\overrightarrow{a}'\overrightarrow{u}'+b}{W} \right\rfloor$. It corresponds to a hash function in the transformed space, where $\overrightarrow{u}' = L\overrightarrow{u}$ is an input object and $\overrightarrow{a}' = \overrightarrow{a}L^{-1}$ the new hyperplane. Lemma 5.1 is proven. □

Locality-Preserving. By the definition of LSH (Sec. 2.2), if the hash functions are locality-preserving, then the LSH index H is effective in hashing the nearby objects into the same buckets based on the learned distance metric D_M. Therefore, if $f'()$ (Eq. 9) is locality-preserving, then the LSH index H built on the Euclidean distance is still effective in supporting the similarity search under D_M.

Next, we show the entries in the hyperplane $\overrightarrow{a}'$ of $f'()$ (Eq. 9) correspond to a normal distribution $N(0, I)$ as long as the eigenvalues of the transform matrix M satisfy certain properties. By [12], this assures that the hash function $f'()$ using $\overrightarrow{a}'$ as the hyperplane is locality-preserving.

Given a hash table produced by h_n hash functions, we use matrix A to represent the hyperplanes of all hash functions. Therefore, A is a $h_n \times d$ matrix, where d denotes the dimension of the data. Each row corresponds to the hyperplane (hash vector) $\overrightarrow{a}$ of one hash function.

Since a valid transform matrix M is always symmetric and positive definite as discussed in Sec. 2.1, it can be decomposed into $M = R^T S R$, where $R^T R = I$ and S is a diagonal matrix formed by the eigenvalues of M. Since $M = L^T L$, we get $L = S^{1/2}R$.

$$A' = AL^{-1} = AR^{-1}S^{-1/2} \tag{10}$$

We use $[\overrightarrow{e_1}, \overrightarrow{e_2}, ..., \overrightarrow{e_d}]$ to represent AR^{-1}. $\overrightarrow{e_i}$ represents the ith column of AR^{-1}. We denote the elements at the diagonal of $S^{-1/2}$ as $[s_1, s_2, ..., s_d]$. We then have $AR^{-1}S^{-1/2} = [s_1\overrightarrow{e_1}, s_2\overrightarrow{e_2}, ..., s_d\overrightarrow{e_d}]$.

Next, we show each row vector in $AR^{-1}S^{-1/2}$ which represents the hyperplane $\overrightarrow{a}'$ of one hash function $f'()$ corresponds to a vector drawn from normal distribution $N(0, I)$ as long as the eigenvalues of the transform matrix M satisfy certain properties. By [12], this assures that the hash function $f'()$ using $\overrightarrow{a}'$ as the hyperplane is locality-preserving.

Lemma 5.2. *Statistically the hyperplane $\overrightarrow{a}'$ in Eq. 9 corresponds to a vector drawn from a normal distribution $N(0, I)$ as long as $\forall\, s_i$ in the main diagonal of $S^{-1/2}$: $0.05 < P(\chi^2_{h_n} < h_n s_i^2) < 0.95$, where p-value is set as 0.1.*

Proof. We first show each row vector in AR^{-1} obeys a normal distribution $N(0, I)$. Since each row vector in A obeys a normal distribution $N(0, I)$, where I is the identity matrix, then each row vector in AR^{-1} obeys a multivariate normal distribution $N(0, \Sigma)$ [18], where $\Sigma = R^{-1}(R^{-1})^T$ is the covariance matrix. Since $R^T R = I$, it is easy to see that $\Sigma = R^{-1}(R^{-1})^T = I$. Thus each row vector in AR^{-1} can be regarded as a vector drawn from a normal distribution $N(0, I)$.

Note this indicates that each element in AR^{-1} can be regarded as a sample drawn from a normal distribution $N(0, 1)$. Therefore, each column vector $\overrightarrow{e_i}$ in AR^{-1} can also be regarded as a vector drawn from a normal distribution $N(0, I)$.

Next, we show each column vector $s_i\overrightarrow{e_i}$ in $AR^{-1}S^{-1/2}$ can also be regarded as a vector drawn from a normal distribution. This will establish our conclusion that each row vector in $AR^{-1}S^{-1/2}$ which represents the hyperplane $\overrightarrow{a}'$ of one hash function $f'()$ can be regarded as a vector drawn from normal distribution [18].

We use χ^2 test to determine if $\forall s_i\overrightarrow{e_i} \in [s_1\overrightarrow{e_1}, s_2\overrightarrow{e_2}, ..., s_d\overrightarrow{e_d}]$ statistically corresponds to a vector drawn from normal distribution $N(0, I)$ [34]. Given a h_n dimension vector drawn from normal distribution $N(0, I)$, the sum of the squares of its h_n elements obeys χ^2 distribution with h_n degrees of freedom [28]. We already know that $\overrightarrow{e_i}$ is a vector drawn from a normal distribution $N(0, I)$. Therefore, as long as the sum of the squares of the elements in $s_i\overrightarrow{e_i}$ also obeys χ^2 distribution with h_n degrees of freedom, statistically $s_i\overrightarrow{e_i}$ corresponds to a vector drawn from a normal distribution $N(0, I)$ [34].

First, we calculate the expected value of the sum of the squares of the elements in $s_i\overrightarrow{e_i}$. Since $\overrightarrow{e_i}$ corresponds to a vector drawn from normal distribution $N(0, I)$, $E(\sum_{j=0}^{h_n} e_{ij}^2) =$

[1] Here we consider $\overrightarrow{a}'$ and $\overrightarrow{a}$ as row vectors for ease of presentation.

h_n, where e_{ij} denotes the jth element in $\overrightarrow{e_i}$. Since s_i is a constant, we get $E(\sum_{j=0}^{h_n}((s_i e_{ij})^2 = s_i^2 E(\sum_{j=0}^{h_n}(e_{ij}^2) = h_n s_i^2$.

We set the p-value as 0.1. By [34], if $0.05 < P(\chi^2_{h_n} < h_n s_i^2) < 0.95$, we cannot reject the hypothesis that $s_i\overrightarrow{e_i}$ is a vector drawn from a normal distribution. This immediately shows that each row vector $\overrightarrow{a}'$ in $AR^{-1}S^{-1/2}$ statistically corresponds to a vector drawn from a normal distribution $N(0, I)$. □

Testing Method. Testing whether Lemma 5.2 holds is straightforward. That is, we first compute the eigenvalues eg_i of the transform matrix M. Since S is a diagonal matrix formed by the eigenvalues of M, the element s_i at the main diagonal of $S^{-1/2}$ corresponds to $eg_i^{-1/2}$. We then use χ^2 table [34] to test if $0.05 < P(\chi_n^2 < h_n s_i^2) < 0.95$. This in fact corresponds to $0.05 < P(\chi_n^2 < h_n eg_i^{-1}) < 0.95$. The eigenvalue computation and χ^2 test can be easily done using Python libraries like Numpy and SciPy [35]. As an example, given an LSH index with 3 hash functions, by [34], ∀ eigenvalue eg_i of M, if $0.35 < 3 \times eg_i^{-1} < 7.81$, that is $0.38 < eg_i < 8.57$, then we do not have to update the LSH index.

Analysis. Now we analyze why Lemma 5.2 is often met.

By Eq. 5, $M_{t+1} = argmin D_{ld}(M_{t+1}, M_t) + \eta l(d_{t+1}, d_t^*)$. $l(d_{t+1}, d_t^*)$ represents the difference between the targeted distance and the distance computed using M_{t+1}. Since many solutions exist that can make $l(d_{t+1}, d_t^*)$ close to 0 [38], the solution of Eq. 5 is mainly determined by $D_{ld}(M_{t+1}, M_t)$.

Initially, $M_t = I$ when t = 0. Accordingly, $D_{ld}(M_{t+1}, M_t)$= $D_{ld}(M_{t+1}, I)$ = $tr(M_{t+1})$ - logdet(M_{t+1}) - d. Note $det(M_{t+1}) = \prod_{i=1}^{d} eg_i$, $tr(M_{t+1}) = \sum_{i=1}^{d} eg_i$, where eg_i denotes the eigenvalue of M_{t+1}. By [45], D_{ld} is minimized when $eg_1 = eg_2 = \ldots = eg_d = \sqrt[d]{det(M_{t+1})}$. Since in distance metric learning, d usually is not small [47], $\sqrt[d]{det(M_{t+1})}$ is close to 1 [45]. In a similar manner, we can show that M_{t+1} has similar eigenvalues to M_t when $t > 0$. By induction, the eigenvalues of M_{t+1} tend to be close to 1, and thus tend to satisfy the requirement in Lemma 5.2.

6 THE UPDATE OF LSH

The LSH index should be updated when the distance metric changes, and Lemma 5.2 is not satisfied. To assure efficiency of similarity search, we now propose a fast LSH update mechanism. It only re-hashes the objects that have a high probability of being moved to a different bucket due to an update of the distance metric. It includes two key innovations: (1) mapping the problem of finding the objects that are strongly impacted by the update of the distance metric to an inner product similarity search problem; and (2) quickly locating such objects using an auxiliary inner product similarity-based LSH index.

Measure the Influence of Distance Metric Change. Given a learned distance metric D_M with transform matrix $M = L^T L$, by Sec. 2.1, computing the distance using the learned distance metric is equivalent to computing Euclidean distance in an input space transformed by L. Therefore, building an LSH index that effectively serves the similarity search using the learned distance metric D_M corresponds to hashing the input objects in the transformed space.

Suppose we have two distance metrics w.r.t. transform matrices $M_{old} = L_{old}^T L_{old}$ and $M_{new} = L_{new}^T L_{new}$, then an LSH index corresponding to each of these two metrics respectively can be built using the following hash functions:

$$f(\overrightarrow{u})_{old} = \left\lfloor \frac{\overrightarrow{a} \cdot L_{old}\overrightarrow{u} + b}{W} \right\rfloor, f(\overrightarrow{u})_{new} = \left\lfloor \frac{\overrightarrow{a} \cdot L_{new}\overrightarrow{u} + b}{W} \right\rfloor$$

Then given an object $\overrightarrow{u}$ as input, the difference between the outputs produced by the two hash functions can be computed as:

$$f(\overrightarrow{u})_{diff} = \left\lfloor \frac{\overrightarrow{a} \cdot (L_{diff}\overrightarrow{u})}{W} \right\rfloor \tag{11}$$

where $L_{diff} = L_{old} - L_{new}$ represents the *delta matrix* between L_{old} and L_{new}. $f(\overrightarrow{u})_{diff}$ represents how much the change of the distance metric impacts $\overrightarrow{u}$ in the LSH index.

Obviously, it is not practical to find the objects that are strongly impacted by the change of the distance metric by computing $f(\overrightarrow{u})_{diff}$ for each input object, since it would be as expensive as re-hashing all objects in the index.

An Inner Product Similarity Search Problem. As shown in Eq. 11, the value of $f(\overrightarrow{u})_{diff}$ is determined by $\overrightarrow{a} \cdot (L_{diff}\overrightarrow{u})$. Since $\overrightarrow{a} \cdot (L_{diff}\overrightarrow{u}) = \overrightarrow{a}^T L_{diff} \cdot \overrightarrow{u}$, the value of $f(\overrightarrow{u})_{diff}$ is thus determined by $\overrightarrow{a}^T L_{diff} \cdot \overrightarrow{u}$, that is, the inner product between $\overrightarrow{a}^T L_{diff}$ and $\overrightarrow{u}$. The larger the inner product is, the larger the $f(\overrightarrow{u})_{diff}$ is. Therefore, the problem of finding the objects $\overrightarrow{u}$ that should be re-hashed can be converted into the problem of finding the objects that have a *large inner product* with a *query object* $\overrightarrow{q} = \overrightarrow{a}^T L_{diff}$, where $\overrightarrow{q}$ is constructed by the hyperplane $\overrightarrow{a}$ used in the hash function and the delta matrix L_{diff}.

By [44], finding the objects $\overrightarrow{u}$ that have a large inner product with the object $\overrightarrow{q}$ corresponds to an inner product similarity search problem. This problem can be efficiently solved if we have an LSH index that effectively supports inner product similarity search. Whenever the LSH has to be updated, to find the objects that have to be re-hashed, we thus only need to construct the query object $\overrightarrow{q}$ and then submit a similarity search query to the inner product-based LSH index.

Next, we first present a data augmentation-based LSH mechanism to efficiently support inner product similarity search. We then introduce a multi-probe based strategy that effectively leverages the output of the inner product similarity search to find the objects that have to be updated.

6.1 Inner Product Similarity LSH

The key insight of the LSH mechanism is that by augmenting the query object $\overrightarrow{q}$ and the input object $\overrightarrow{u}$ with one extra dimension, the inner product similarity search can be efficiently supported by leveraging *any* classical cosine similarity-based LSH.

First, consider the cosine similarity defined in [43] as:

$$cos(\overrightarrow{q}, \overrightarrow{u}) = \frac{\overrightarrow{q} \cdot \overrightarrow{u}}{\left\|\overrightarrow{q}\right\| \left\|\overrightarrow{u}\right\|} \tag{12}$$

The inner product similarity [36, 42] can be expressed as:

$$inner(\overrightarrow{q}, \overrightarrow{u}) = \overrightarrow{q} \cdot \overrightarrow{u} = \frac{\overrightarrow{q} \cdot \overrightarrow{u}}{\left\|\overrightarrow{q}\right\| \left\|\overrightarrow{u}\right\|} \times \left\|\overrightarrow{q}\right\| \times \left\|\overrightarrow{u}\right\| \tag{13}$$

By Eq. 12 and 13, we get:

$$inner(\overrightarrow{q}, \overrightarrow{u}) = cos(\overrightarrow{q}, \overrightarrow{u}) \times \left\|\overrightarrow{q}\right\| \times \left\|\overrightarrow{u}\right\| \tag{14}$$

In our OASIS, the inner product similarity search is used to search for the neighbors of the query object $\overrightarrow{q}$. During one similarity search, Eq. 14 is repeatedly used to compute the distance between $\overrightarrow{q}$ and different input objects $\overrightarrow{u}$. Therefore, for our similarity search purpose, $\left\|\overrightarrow{q}\right\|$ can be safely ignored from Eq. 14, because it does not influence the relative affinity from different $\overrightarrow{u}$ to $\overrightarrow{q}$.

However, the search results are influenced by $\left\|\overrightarrow{u}\right\|$. Even if an object $\overrightarrow{u}$ has a high cosine similarity with the query object $\overrightarrow{q}$, it does not necessarily mean that $\overrightarrow{u}$ has a high inner product similarity with $\overrightarrow{q}$ because of varying $\left\|\overrightarrow{u}\right\|$ of different $\overrightarrow{u}$.

We now propose to fill this gap between the inner product similarity and cosine similarity by appending one extra dimension to the query object $\overrightarrow{q}$ and the input object $\overrightarrow{u}$ as shown below:

$$\overrightarrow{q}' = \left[\overrightarrow{q}, 0\right], \overrightarrow{u}' = \left[\overrightarrow{u}, \sqrt{1 - \left\|\overrightarrow{u}\right\|^2}\right]$$

Next, we show that after this data augmentation, the inner product similarity $inner(\overrightarrow{q}, \overrightarrow{u})$ can be computed based on the cosine similarity between the augmented $\overrightarrow{q}'$ and $\overrightarrow{u}'$.

Lemma 6.1.

$$inner(\overrightarrow{q}, \overrightarrow{u}) = \overrightarrow{q} \cdot \overrightarrow{u} = cos(\overrightarrow{q}', \overrightarrow{u}') \times \left\|\overrightarrow{q}\right\| \tag{15}$$

Proof. Using $\overrightarrow{q}'$ and $\overrightarrow{u}'$ in the computation of Cosine similarity, we get $cos(\overrightarrow{q}', \overrightarrow{u}') = \frac{\overrightarrow{q} \cdot \overrightarrow{u}}{\left\|\overrightarrow{q}\right\| \left\|\overrightarrow{u}'\right\|}$. Since $\left\|\overrightarrow{u}'\right\| = \left\|\left[\overrightarrow{u}, \sqrt{1 - \left\|\overrightarrow{u}\right\|^2}\right]\right\| = 1$, we get $cos(\overrightarrow{q}', \overrightarrow{u}') = \frac{\overrightarrow{q} \cdot \overrightarrow{u}}{\left\|\overrightarrow{q}\right\|}$. In turn, we get $inner(\overrightarrow{q}, \overrightarrow{u}) = \overrightarrow{q} \cdot \overrightarrow{u} = cos(\overrightarrow{q}', \overrightarrow{u}') \times \left\|\overrightarrow{q}\right\|$. □

By Lemma 6.1, if we build a cosine similarity LSH index for the augmented objects, then the original objects that fall into the same bucket will have a high inner product similarity.

Therefore, our data augmentation strategy ensures that any cosine similarity LSH index can be effectively used to support inner product similarity search. In this work, we adopt the method introduced in [2].

6.2 Cosine Similarity LSH-based Update

Accuracy VS Efficiency. OASIS re-hashes the objects that have a large inner product with the query object $\overrightarrow{q}$. These objects are produced using a similarity query conducted on the inner product LSH introduced above. Therefore, the cost of adapting the LSH to fit the new distance metric depends on the number of objects returned by the similarity query. If too many objects were to be returned, then the update would be expensive. On the other hand, returning too few objects tends to miss the objects that should have changed their buckets in the new LSH. Hence it risks to sacrifice its accuracy in supporting similarity search when using the new distance metric.

In this work, we tackle this delicate trade off by designing a multi-probe LSH [31] based approach. This approach returns as few objects as possible that could potentially change their bucket membership in the new LSH, while still ensuring the accuracy of the new LSH in supporting similarity search.

Multi-probe LSH. As shown in Fig. 3, compared to the traditional LSH solution, a multi-probe LSH [31] divides each large bucket into m small buckets. Then at query time, instead of probing one single bucket b that the query object falls into, it probes the m small buckets. Multi-probe tends to have a higher accuracy for the nearest neighbor search task than the traditional LSH [31].

As an example, in Fig. 3, the traditional LSH returns n_1 and n_3 as the k nearest neighbors of $\overrightarrow{q}$ ($k = 2$ in this case). However, in fact, n_2 is closer to $\overrightarrow{q}$ than n_3. In this case, n_2 is missed, because it falls into a nearby bucket. Multi-probe LSH (in this case $m = 3$) instead correctly returns n_1 and n_2 as nearest neighbors of $\overrightarrow{q}$, because it also probes the two neighboring buckets. As can be observed, the larger m is, the higher the accuracy of the nearest neighbor search becomes. However, a larger m also increases the search costs.

Avoidance of Additional Error in kNN Search. The observation here is that using multi-probe, even if we do not re-hash an object that should be moved to a different bucket from the old LSH to the new LSH w.r.t. the new learned distance metric, often it does not impact the accuracy of kNN search using the new distance metric, as shown below.

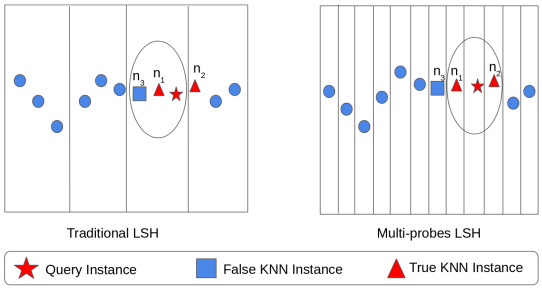


Figure 3: Multi-probe VS traditional LSH

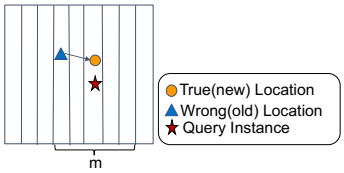


Figure 4: Avoidance of Additional Error in kNN Search

Given one input object $\overrightarrow{u}$, assume it is hashed into bucket b_{old} in the old LSH index H, while it should have been hashed into bucket b_{new} in the new LSH H'. Therefore, $\overrightarrow{u}$ moves $e = \mid b_{old} - b_{new} \mid$ buckets due to the change of the distance metric, where e can be computed using Eq. 11. For example, in Fig. 4, object $\overrightarrow{u}$ moves two buckets. Suppose $\overrightarrow{u}$ is a nearest neighbor of query $\overrightarrow{q}$. $\overrightarrow{q}$ should be hashed into bucket b_{new} in H' per the property of LSH. Suppose $\overrightarrow{u}$ is not re-hashed and thus remains in bucket b_{old} even in the new LSH H'. In this case, if $m \geq 2 \times e + 1$, then the multi-probe strategy is still able to find $\overrightarrow{u}$ as the nearest neighbor of $\overrightarrow{q}$. As shown in Fig. 4, if we set m as $5 = 2 \times 2 + 1$, then $\overrightarrow{u}$ will still be correctly captured as the nearest neighbor of $\overrightarrow{q}$, because multi-probe also considers the objects in the 2 nearby buckets left to b_{new} as candidates.

Search the To-be-updated objects. Based on this observation, we design a search strategy that by only examining the objects in a small number of buckets, efficiently locates the objects that could introduce additional error in the approximate kNN search if not re-hashed in the new LSH.

As shown in Alg. 2, given the delta matrix $L_{diff} = L_{new} - L_{old}$ and the parameter m of multi-probe, we first construct the query object $\overrightarrow{q}$ as $\overrightarrow{a}^T L_{diff}$ (Line 2). Then we search for the to-be-re-hashed objects starting from the bucket b of the inner product LSH that $\overrightarrow{q}$ falls into. For each object $\overrightarrow{u}$ in bucket b, using Eq. 11 we compute the number of buckets that $\overrightarrow{u}$ moves denoted as e (Lines 7-9). $\overrightarrow{u}$ will be re-hashed if e is larger than $\frac{m-1}{2}$ (Lines 10-11). Otherwise, additional error may be introduced in the kNN search. If the e averaged over all objects in bucket b, denoted as $avg(e)$, is larger than $\frac{m-1}{2}$, the buckets b_n nearby b are searched iteratively until the $avg(e)$ of a bucket b_n is under $\frac{m-1}{2}$ (Lines 13-17). These objects are then re-hashed using $f(\overrightarrow{u})_{new} = \left\lfloor \frac{\overrightarrow{a} \cdot L_{new} \overrightarrow{u} + b}{W} \right\rfloor$.

Algorithm 2 Locate to-be-updated Objects

```
1:  function locateUpdate(Delta matrix L_diff, numberOfProbe m, Hash
    vector a, Hash const W)
2:      q = a^T · L_diff;
3:      list[] candidates = null; i = 1;
4:      list[] buckets = buckets.add (H(q));
5:      while (i ≤ (m-1)/2) do
6:          eSum = 0; count = 0;
7:          for each bucket b in buckets do
8:              for each object u in bucket b do
9:                  e = ⌊(a·(L_diff u)) / W⌋;
10:                 if e > (m-1)/2 then
11:                     candidates.add(e);
12:                 eSum +=e; count++;
13:         avg(e) = eSum/count;
14:         if avg(e) > (m-1)/2 then
15:             buckets.addNeighbor(H(q),i); i++;
16:         else
17:             return candidates;
```

7 EXPERIMENTAL EVALUATION

7.1 Experimental Setup and Methodologies

Experimental Setup. All experiments are conducted on a LINUX server with 12 CPU cores and 512 GB RAM. All code is developed in Python 2.7.5.

Datasets. We evaluate our OASIS using both real datasets and synthetic datasets.

Real Datasets. We use four real datasets **publicly available** with various of dimensions and sizes, *PageBlock* [33], *KDD99* [39], CIFAR10 [26], and *HEPMASS*, to evaluate how OASIS performs on different analytics tasks.

PageBLOCK contains information about different types of blocks in document pages. There are 10 attributes in each instance. PageBlock contains 5,473 instances, with 560 instances that are not text blocks labeled as class 1 and the other text instances labeled as class 2. PageBlock is used to evaluate the performance of OASIS for kNN classification.

CIFAR10 [26] is a popular benchmark dataset for image classification. It includes 10 classes and 60,000 training images. All images are of the size 32×32. Following the typical practice [14], we first use unsupervised deep learning to extract a 4,096 dimensional feature vector to represent each raw image. We then perform PCA to further reduce the dimension of the feature vectors to 100 dimensions. CIFAR10 is used in our labeling evaluation.

KDD99 corresponds to network intrusion detection data curated as benchmark data set for outlier detection study. There are 24 classes of intrusion connections and normal connections. It contains in total 102,053 instances with 3,965 intrusion connections labeled as outliers. Each instance has 35 attributes plus the class label. Given KDD99 is an outlier benchmark, we use it to evaluate the effectiveness of OASIS in outlier detection.

HEPMASS was generated in high-energy physics experiments for the task of searching for the signatures of exotic particles. There are 10,500,000 instances in HEPMASS. Each instance has 28 attributes. It is composed of two classes of instances. 50% of the instances correspond to a signal process. The other instances are a background process.

Synthetic Datasets. Since most public data sets with labels are relatively small, we use synthetic datasets to evaluate the scalability of OASIS, in particular the performance of our LSH update strategy. We generate synthetic datasets using scikit-learn 0.19.1. Each synthetic dataset has 50 attributes. The largest dataset contains 100 million instances. The instances are divided into 2 classes with 50 million instances each. The data in each class is skewed and can be grouped into multiple clusters.

Experimental Methodology & Metrics. We evaluate the **effectiveness** and **efficiency** of our OASIS approach in four different data analytics tasks, including labeling, kNN classification, outlier detection, and data exploration supported by kNN queries.

Effectiveness. When evaluating the kNN classification, we measure the accuracy of the prediction. When evaluating the outlier detection, we measure the F1 score. When evaluating the labeling, we measure the accuracy of the automatically produced labels. When evaluating the kNN query, we measure the accuracy of the returned results.

We compare OASIS against three alternative approaches: (1) baseline approach (*No Update*) that does not update the distance metric; (2) static update approach (*Static Update*) that re-learns the transform matrix from scratch each time when a new example is collected. It repeatedly applies the most popular distance metric learning methods with respect to different tasks [47, 49, 51]; It is expected to produce the distance metric **best fitting** the similarity examples seen over time; (3) an online metric learning method *LEGO* [24]

Efficiency. Efficiency is evaluated by the running time. We measure the running time of our continuous metric learning method and our LSH update method separately.

Same to the effectiveness evaluation, we compare the efficiency of our incremental metric learning method in OASIS against (1) No Update, (2) Static Update, and (3) LEGO.

For evaluating our LSH update method in OASIS, we measure the total running time of similarity searches plus the update of the LSH index. We compare against two alternative approaches: (1) similarity search by *linear scan* without the support of LSH (*Linear Scan*); and (2) similarity search with the LSH rebuilt from scratch whenever the distance metric changes (*Rebuild LSH*). We use the large HEPMASS dataset and synthetic datasets to evaluate the LSH update method of OASIS. The other three real datasets are small and thus not appropriate for this scale measurement experiment.

Evaluation of the Proposed Inner Product LSH. We also evaluate our proposed inner product LSH by measuring the recall and query time.

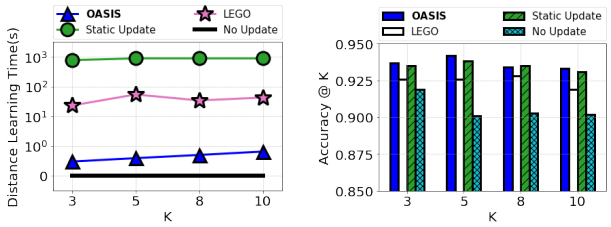
7.2 Evaluation of kNN Classification

When supporting kNN classification, OASIS starts by using Euclidean distance as the initial distance metric. If a testing object is misclassified, it is used as a new example that triggers the update process of OASIS, namely updating the distance metric model and the LSH. *Static Update* re-learns the distance metric using LMNN [47] – the most popular distance metric learning method used in kNN classifier.

Evaluation of Classification Accuracy. We evaluate how OASIS improves the accuracy of kNN classification using the *PageBlock* data set. Initially, the labeled training set contains 1,000 instances. We vary the parameter K, the most important parameter in kNN classifier, from 3 to 10.

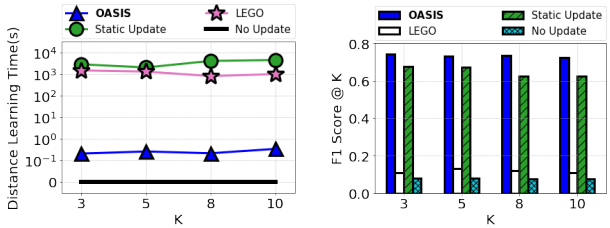
As shown in Fig. 5(b), OASIS outperforms *No Update* in all cases because of the progressively enriched labeled dataset and the continuous update of the distance metric model, while *No Update* always uses the initially labeled dataset and the fixed distance metric model which becomes stale over time. Further, the accuracy of OASIS is very close to *Static Update*. Static Update re-learns the distance metric from scratch every time when a new example is acquired, therefore its performance is expected to be the best. This shows that the distance metric model incrementally learned by OASIS is close to the optimal model, while being much more efficient than *Static Update* (see Fig. 5(a)). LEGO also incrementally updates the distance metric model. However, it performs worse than OASIS in all cases, although better than No Update. This is as expected, because when forming the new distance constraint for a new received similarity example, LEGO first computes the pairwise distances among all similarity examples received so far and uses the 5th percentile distance as the target distance. Using a target distance computed by this hard-coded rule, given one particular similarity example, its neighbors in the same class are not necessarily closer than its non-matching neighbors, potentially introducing classification errors in LEGO. In contrast, OASIS computes a customized target distance for each example that ensures this similarity example will not get mis-classified any more, as described in Sec. 4.1.1.

Evaluation of Efficiency. As shown in Fig. 5, OASIS significantly outperforms Static Update up to 3 orders of magnitude in updating the distance model. This is because the incremental update strategy has a time complexity that is quadratic in the dimensionality of the data and is not dependent on the number of examples acquired so far. While Batch Update and Static Update both have to re-learn the distance metric model from scratch. Apart from the time complexity which



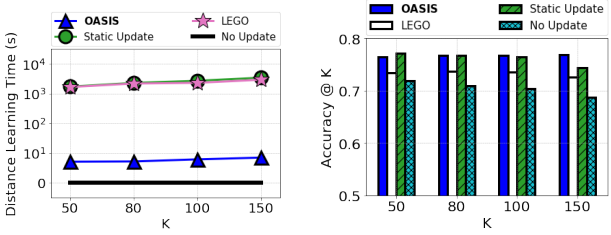
(a) CPU time: varying K (b) Accuracy: varying K

Figure 5: KNN Classification: Accuracy and CPU Time (PageBLOCK)



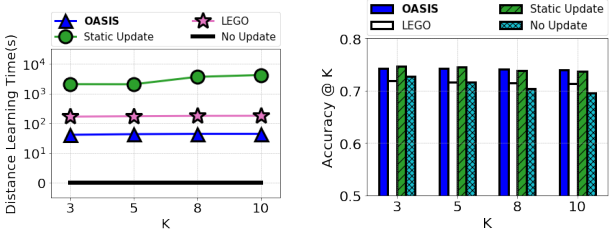
(a) CPU time: varying K (b) F1 Score: varying K

Figure 6: Outlier Detection: F1 score and CPU Time (KDD99)



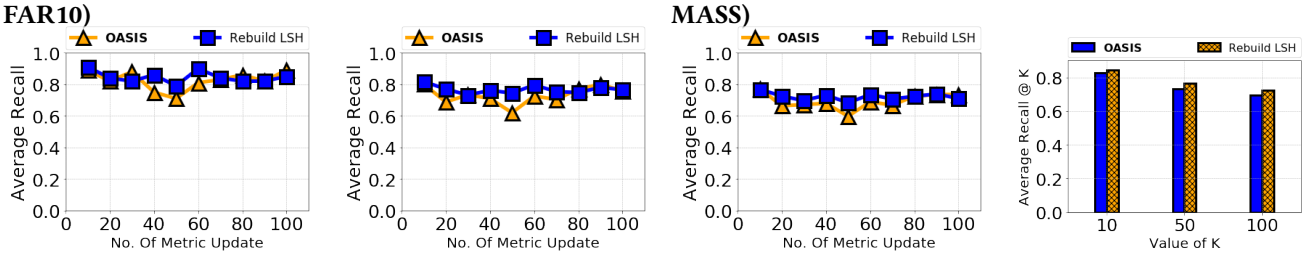
(a) CPU time: varying K (b) Accuracy: varying K

Figure 7: Labeling: Accuracy and CPU Time (CIFAR10)



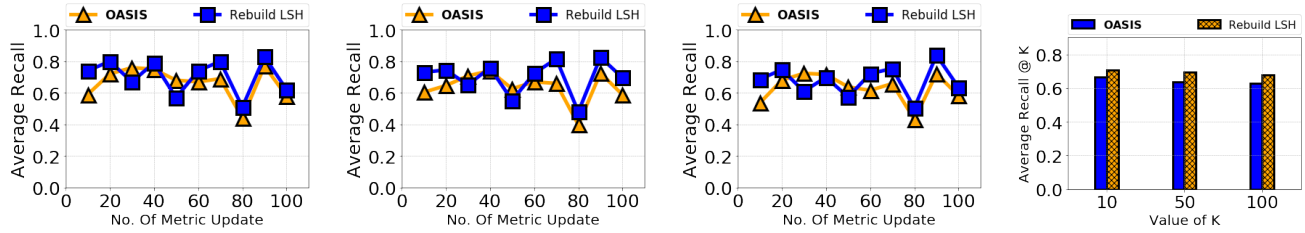
(a) CPU time: varying K (b) Accuracy: varying K

Figure 8: KNN Query: Accuracy and CPU Time (HEPMASS)



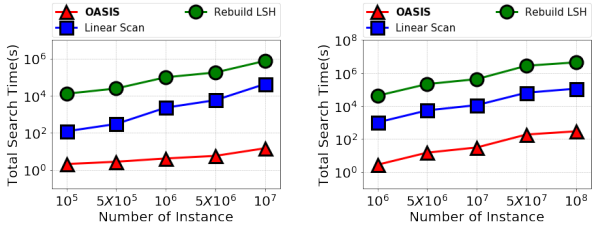
(a) k=10 (b) k=50 (c) k=100 (d) Average

Figure 9: Recall metric for KNN query with LSH udpate: Varying K (HEPMASS dataset)



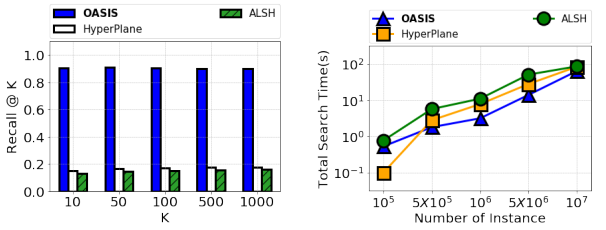
(a) k=10 (b) k=50 (c) k=100 (d) Average

Figure 10: Recall metric for KNN query with LSH update: Varying K (Synthetic dataset)



(a) HEPMASS dataset (b) Synthetic dataset

Figure 11: CPU time of LSH update with varying dataset sizes



(a) Recall: Varying value of K (b) CPU Time: Varying Dataset Size

Figure 12: Inner Product LSH: varying K (HEPMASS)

is quadratic in the size of the labeled dataset and in number of data dimensions, learning the distance metric model from scratch also consumes a huge amount of memory. Therefore, it tends to fail due to an out of memory error when the number of examples is large. Our OASIS instead requires near-zero extra memory besides the transform matrix.

OASIS is up to 100x faster than LEGO in updating the distance metric. This is because LEGO has to be aware of all pairwise distances among all examples seen so far in each update, while OASIS only needs the kNN to the new example to produce the distance constraints.

7.3 Evaluation of Outlier Detection

We use the KDD99 dataset to evaluate the performance of OASIS on supporting distance-based outliers. If an object is mis-classified, we use the strategy discussed in Sec. 4.1.2 to produce the distance constraints and then update the distance metric. For Static Update, we insert the newly acquired outlier or inlier examples into the training set and re-learn the distance metric using MELODY [51] – the most recent distance metric learning-based outlier detection method. Initially the training set has 1000 examples. Similar to our OASIS, LEGO also online updates the distance metric when an object gets mis-classified, using the constraints produced by its own strategy.

Evaluation of F1 Score. We evaluate how OASIS improves the accuracy of outlier detection by continuously updating the distance metric. The input parameter K is varied in this case. As shown in Fig. 6(b), OASIS outperforms *Static Update* in F1 score by about 10%. Note since *Static Update* re-learns the distance metric from scratch each time when a new example is acquired, we expected it to perform the best. However, this is not the case. This indicates that the general incremental update strategy used in OASIS is very effective, sometimes even better than the distance metric learning method customized to the analytics tasks. The F1 scores of *No Update* and *LEGO* are much lower. The reason that LEGO performs poorly is that it uses the 5th percentile pairwise distance as target distance. The learned distance metric using this small target distance tends to make the outliers close to their neighbors. This is not suitable for outlier detection, since based on the general principle of outlier detection, outliers correspond to the objects that have large distances to their neighbors [15].

Evaluation of Efficiency. As shown in Fig. 6(a), OASIS is up to 5 orders of magnitude faster than *Static Update* and 3 orders of magnitude faster than the online algorithm *LEGO*.

7.4 Evaluation of Labeling

We use the CIFAR10 dataset to evaluate the effectiveness of OASIS at supporting labeling applications. We randomly select 100 images as an initial labeled data set to learn a distance metric using *LMNN* [47]. Then we randomly select 3,000 images as labeled query set and consider the rest of CIFAR10 as an unlabeled set. We then propagate labels to the unlabeled set using kNN queries following the OASIS procedure described in Sec. 1.

Evaluation of Accuracy. The labeling accuracy is measured as the ratio of the number of correctly labeled objects to all labeled objects by propagation. As shown in Sec. 7(b), the accuracy of OASIS is comparable to the *Static Update* solution, and higher than No-Update and LEGO. Note when k increases, the accuracy of system increases even further. This property is particularly useful in labeling, because a larger k indicates that more labels can be automatically produced.

Evaluation of Efficiency. As shown in Fig. 7(a), OASIS is 2 orders faster than *Static Update* and *LEGO*. The processing times of Static Update and LEGO are close to each other.

7.5 Evaluation of kNN Queries

The results are shown in Fig. 8. Please refer to the extended version [3] of this manuscript for the analysis.

7.6 Evaluation of LSH Updates

We use the largest real dataset HEPMASS and large synthetic datasets to evaluate both the effectiveness and efficiency of our LSH update method. For effectiveness, we measure the recall of the exact kNN among the candidates returned from LSH. For efficiency, we measure the overall processing time including the time for updating LSH and for kNN search. 100 distance metric updates are triggered in each case. Each LSH index uses 3 hash tables.

Effectiveness. The recall is measured for 100 queries. We vary the k parameter (k = 10, 50 and 100) in the kNN queries. We compare the LSH update method in OASIS against *Rebuild* which re-hashes all objects whenever the distance metric changes.

Fig. 9 and 10 show how the recall of our LSH update method and *Rebuild* change over time during the update process. The two curves w.r.t. LSH update and Rebuild look similar. In addition, we also measure the average recall. As shown in Figs. 9(d) and 10(d), the average recall of LSH update is only slightly lower than *Rebuild*, confirming the effectiveness of our LSH update strategy.

Efficiency. We compare our LSH update with *Linear Scan* without using indexing and *Rebuild LSH*. We measure the total processing time of 1,000 kNN queries on each dataset.

Real HEPMASS dataset. We vary the sizes of the subsets of HEPMASS used in the evaluation from 100k to 10M. As shown in Fig. 11(a), our LSH update method consistently outperforms *Rebuild LSH* by 3 to 4 orders of magnitude. This is because our LSH update strategy avoids the update of the LSH index in most cases. Further, in the rare case of an actual update, only a small number of objects are re-hashed. Our

LSH update method also consistently outperforms *Linear Scan* by 2 to 3 orders of magnitude, because the time complexity of kNN search is reduced from linear to sub-linear for LSH. Rebuild LSH is even slower than Linear Scan, although it also uses the sub-linear time complexity approximate kNN search. This is due to the costs of having to re-hash all objects in each update. This update overhead outweighs the query time saved by the approximate kNN search.

Synthetic Datasets. The results on the synthetic datasets are similar to those of the HEPMASS dataset as shown in Fig. 11(b). When the dataset gets larger to 100M, our method outperforms the alternative methods even more. This confirms that our method is scalable to large datasets.

7.7 Evaluation of Inner Product LSH

We compare our inner product LSH used in OASIS to quickly find the to-be-rehashed objects against two state-of-the-art inner product LSH methods ALSH [42] and Hyperplane [36] using the real HEPMASS dataset.

Following the typical practice of evaluating approximate similarity search [29], we measure both recall and query processing time. For this, we configure all LSH methods to have a similar response time by first tuning the parameters on a small sample dataset (10k) and then evaluating the accuracy on larger datasets. As confirmed in Fig. 12(b), after parameter tuning, these methods indeed have a similar query time when increasing the sizes of the datasets (with our OASIS slightly faster).

As shown in Fig 12(a), the OASIS inner product LSH significantly outperforms the alternate methods ALSH and Hyperplane. When varying the parameter K, the recall of OASIS LSH is consistently higher than 0.85, while the recall of ALSH and Hyperplane remain below 0.2. This is because our data augmentation mechanism ensures that inner product similarity search can be supported using the cosine similarity LSH, while cosine similarity LSH is well studied in the literature and achieves high recall [2].

8 RELATED WORK

Online Learning. Some prior research [9, 24, 41] focused on online metric learning. [24] solves for the optimization function using the exact gradient. This solves the problem that traditional online learning approaches could produce non-definite matrices [13] because of the approximate solution for the gradient of the loss function. However, [24] cannot ensure that the previously mis-classified objects are no longer mis-classified in the new transformed space, because it uses the 5th percentile of the pairwise distances among all training examples to set the target distance. We solve this problem by forming the new constraints on the optimization function that take the distance relationship between the matching neighbors of the new item and its non-matching neighbors into consideration.

[9] proposed a method that incrementally learns a good function to measure the similarity between images. [41] focuses on sparse data like image datasets similar to [9]. Both [9] and [41] cannot guarantee the transition matrix M to be positive semidefinite. Therefore, they cannot be directly applied to distance metric learning.

Incremental LSH Update. In [24], a method is sketched to support the update of *cosine similarity*-based LSH in a linearly transformed space. It repeatedly samples some objects to re-hash whenever the transform changes and periodically rebuild the LSH from scratch to ensure its effectiveness. In this work, we prove that the LSH built in the original input feature space is equally effective in supporting similarity search in the transformed input feature space as long as the transform matrix satisfies Lemma 5.2. Therefore, OASIS avoids updating LSH in most cases.

LSH for Inner Product Similarity Search. Maximum Inner product search (MIPS) attracts much attention recently because of its application in Deep Learning. Accordingly, inner Product similarity-based LSH methods were proposed to speed up MIPS [20, 36, 42]. To support MISP, [42] extend the LSH framework to allow asymmetric hashing. [36] instead presents a simple symmetric LSH exists that has stronger guarantees and better empirical performance than the asymmetric LSH in supporting MIPS. However, the LSH in [36], simply using the hyperplane hash function, is shown to have a long running time – often as long as the full linear scan.

Active Distance Metric Learning. In [27, 50], active learning-based approaches were proposed that learn a distance metric by selectively sampling the to-be-labeled objects from the *existing dataset*. Therefore, their focus is on reducing the labeling work from the users. However, unlike our work, they do not account for the continuously incoming objects that had previously not been seen. Therefore, once a distance metric is learned, it will never be updated. We instead focus on the continuous update of the distance metric driven by dynamically acquiring similarity constraints.

9 CONCLUSION

In this work, we propose OASIS that consistently improves the accuracy and efficiency of similarity search-based data mining tasks. OASIS features an incremental distance metric learning strategy that adapts the distance metric whenever new similarity example becomes available. OASIS also offers an update mechanism that only updates the LSH index when it is absolutely necessary and re-hashes a small subset of objects when the update actually happens. Our experimental study confirms the 3 orders of magnitude speedup of our strategies over the state-of-the-art methods.

REFERENCES

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. VLDB, 2003.

[2] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. Practical and optimal lsh for angular distance. In *NIPS*, pages 1225–1233, 2015.

[3] Anonymous. Continuously adaptive similarity search. https://drive.google.com/file/d/1hFsqVD6LlQPRm7SBydk1TRlzGuD1yvf2/view?usp=sharing, 2019.

[4] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD*, pages 29–38, 2003.

[5] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *CoRR*, abs/1306.6709, 2013.

[6] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *SIGMOD*, pages 93–104, 2000.

[7] L. Cao, J. Wang, and E. A. Rundensteiner. Sharing-aware outlier analytics over high-volume data streams. In *SIGMOD*, pages 527–540, 2016.

[8] L. Cao, D. Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner. Scalable distance-based outlier detection over high-volume data streams. In *ICDE*, pages 76–87, 2014.

[9] G. Chechik, U. Shalit, V. Sharma, and S. Bengio. An online algorithm for large scale image similarity learning. In *NIPS*, pages 306–314, 2009.

[10] A. Cichocki, S. Cruces, and S. ichi Amari. Log-determinant divergences revisited: Alpha-beta and gamma log-det divergences. *Entropy*, 17:2988–3034, 2015.

[11] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27.

[12] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, pages 253–262. ACM, 2004.

[13] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. ICML, pages 209–216, New York, NY, USA, 2007. ACM.

[14] A. Dosovitskiy, J. T. Springenberg, M. A. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *NIPS*, pages 766–774, 2014.

[15] C. C. A. S. Edition. *Outlier Analysis*. Springer, 2017.

[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. KDD'96, pages 226–231, 1996.

[17] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive hashing scheme based on dynamic collision counting. SIGMOD, pages 541–552, 2012.

[18] N. Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[19] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[20] Q. Huang, G. Ma, J. Feng, Q. Fang, and A. K. Tung. Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search. In *KDD*, pages 1561–1570. ACM, 2018.

[21] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *SIGMOD*, pages 277–281, 2015.

[22] S. B. Imandoust and M. Bolandraftar. Application of k-nearest neighbor (knn) approach for predicting economic events : Theoretical background. 2013.

[23] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. STOC, pages 604–613. ACM, 1998.

[24] P. Jain, B. Kulis, I. S. Dhillon, and K. Grauman. Online metric learning and fast similarity search. NIPS, pages 761–768, USA, 2008.

[25] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, pages 392–403, 1998.

[26] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[27] K. Kumaran, D. Papageorgiou, Y. Chang, M. Li, and M. Takác. Active metric learning for supervised classification. *CoRR*, abs/1803.10647, 2018.

[28] H. O. Lancaster and E. Seneta. Chi-square distribution. *Encyclopedia of biostatistics*, 2, 2005.

[29] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement. *TKDE*, 2019.

[30] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Dec. 2007.

[31] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. PVLDB, pages 950–961, 2007.

[32] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Intelligent probing for locality sensitive hashing: Multi-probe lsh and beyond. *Proc. VLDB Endow.*, 10(12):2021–2024, Aug. 2017.

[33] D. Malerba, F. Esposito, and G. Semeraro. A further comparison of simplification methods for decision-tree induction. In *Learning from data*, pages 365–374. Springer, 1996.

[34] J. McDonald. *Chi-square test of goodness-of-fit: Handbook of Biological Statistics*. Sparky House Publishing, 2009.

[35] K. Millman and M. Aivazis. Python for scientists and engineers. *Computing in Science & Engineering*, 13(02):9–12, mar 2011.

[36] B. Neyshabur and N. Srebro. On symmetric and asymmetric lshs for inner product search. ICML, pages 1926–1934. JMLR.org, 2015.

[37] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. LOCI: fast outlier detection using the local corr. integral. In *ICDE*, pages 315–326, 2003.

[38] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[39] X. Qin, L. Cao, E. A. Rundensteiner, and S. Madden. Scalable kernel density estimation-based local outlier detection over large data streams. 2019.

[40] A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.

[41] S. Shalev-Shwartz, Y. Singer, and A. Y. Ng. Online and batch learning of pseudo-metrics. In *ICML*, page 94, 2004.

[42] A. Shrivastava and P. Li. Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS). pages 812–821, 2015.

[43] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[44] R. Spring and A. Shrivastava. Scalable and sustainable deep learning via randomized hashing. In *SIGKDD*, pages 445–454, 2017.

[45] J. M. Steele. *The Cauchy-Schwarz master class: an introduction to the art of mathematical inequalities*. Cambridge University Press, 2004.

[46] F. Wang and J. Sun. Survey on distance metric learning and dimensionality reduction in data mining. *Data Min. Knowl. Discov.*, 29(2):534–564, Mar. 2015.

[47] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, pages 1473–1480, 2006.

[48] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244, June 2009.

[49] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *NIPS*,

NIPS'02, pages 521–528, Cambridge, MA, USA, 2002. MIT Press.

[50] L. Yang, R. Jin, and R. Sukthankar. Bayesian active distance metric learning. In *UAI*, pages 442–449, 2007.

[51] G. Zheng, S. L. Brantley, T. Lauvaux, and Z. Li. Contextual spatial outlier detection with metric learning. In *SIGKDD*, pages 2161–2170, 2017.