SAND2016-11619C

Enhanced Approximate Nearest Neighbor via Local Area Focused Search

Antonio Gonzales and Nicholas Blazier
Sandia National Laboratories
PO Box 5800 Albuquerque, NM 87185
aigonza@sandia.gov npblazi@sandia.gov

Abstract

Approximate Nearest Neighbor (ANN) algorithms are increasingly important in machine learning, data mining, and image processing applications. There is a large family of space-partitioning ANN algorithms, such as randomized KD-Trees, that work well in practice but are limited by an exponential increase in similarity comparisons required to optimize recall. Additionally, they only support a small set of similarity metrics. We present Local Area Focused Search (LAFS), a method that enhances the way queries are performed using an existing ANN index. Instead of a single query, LAFS performs a number of smaller (fewer similarity comparisons) queries and focuses on a local neighborhood which is refined as candidates are identified. We show that our technique improves performance on several well known datasets and is easily extended to general similarity metrics using kernel projection techniques.

1. Introduction

Nearest neighbor search is a fundamental algorithm that returns the most similar members of a dataset for a query based on a specified distance metric. It has been applied to a wide range of applications in image processing, pattern recognition, and data retrieval [1]. In practice nearest neighbor search becomes expensive and often impractical as the size of datasets grow larger. To address this scaling research has focused on creating approximate methods for nearest neighbor search [2][11][9][3][10].

To date a number of different approaches have been proposed for Approximate Nearest Neighbor (ANN) search. These approaches build and populate a data structure that allow for fast queries while only failing to find a small percentage of the true nearest neighbors (high recall). This data structure is called an index. *Space partitioning algorithms* aim to divide the search space for fast retrieval. These methods include KD-Trees, ball trees, and hierarchical K-Means trees [2][9]. They tend to be fast to construct and search but can be limited by poor partitioning and in the similarity metrics they support. Another popular approach is to work in the domain of *hash functions* such as Locality-Sensitive Hashing (LSH) [9]. These methods provide constant time queries but are limited by the fact that creating a hash function for a specific dataset can be a difficult problem. *K-Nearest Neighbor graph* is an approach that attempts to build a graph of the entire dataset where edges represent neighbors [3][10]. These methods work well in practice but are limited by their high construction cost. These examples represent just a fraction of the numerous methods that have been developed to build efficient ANN search indexes. Conversely, there have been relatively few techniques developed that focus on leveraging standard ANN indices and improving the way queries are performed.

In this paper we introduce the Local Area Focused Search (LAFS). LAFS takes advantage of locality by searching neighbors of intermediate query results produced by a traditional index. This allows LAFS to search in the real similarity space which greatly improves speed and performance for kernel projected data.

2. Proposed Method

2.1. Motivation

Space-partitioning approximate nearest neighbor indices rely on internal mechanisms to return multiple candidate neighbors [9]. While they vary in construction, these mechanisms typically track a set of potential paths not yet visited in the dataset. These paths are visited in order of likelihood that they will lead to a close neighbor of the query. The challenge is that these likelihoods are approximate and can lead to either duplicate neighbors or candidates with low similarities to the query being identified. Figure 1 illustrates this behavior by examining the number of similarity calculations required to achieve a desired recall using a randomized KD-Tree index. A query tends to find the majority of the true nearest neighbors within the first few hundred candidates it evaluates. Finding the remaining true nearest neighbors requires examining an exponentially increasing number of candidates.

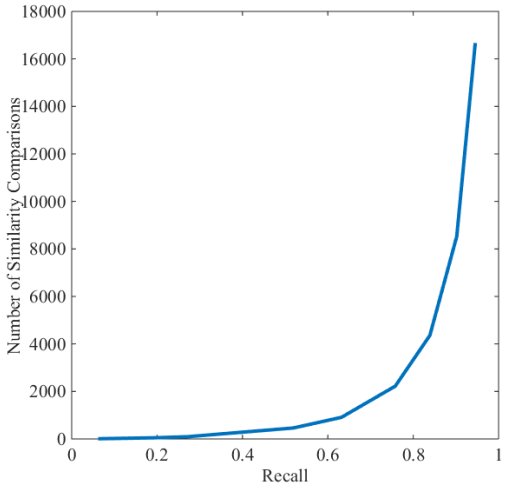

Figure 1. Relationship of the number of similarity calculations to recall for the top 10 nearest neighbors in the SIFT1M dataset indexed by a randomized KD-Tree.

Instead of seeking to return all candidates with one query, our LAFS method combines the results of several small queries (low number of returned candidate neighbors) which are performed with similar neighbors. This exploits the strength of the ANN search while improving the recall for the same number of similarity calculations. LAFS is detailed in following section.

2.2. Local Area Focused Search

Our method is inspired by the K-NN graph work of Dong *et al*. [3] and their principle that "a neighbor of a neighbor is also likely to be a neighbor". While their focus is improved approximation of K-NN Graphs, LAFS, as shown in Algorithm 1, applies this principle to improve the results of a query using a standard ANN index. The concept is that the candidate nearest neighbors which are returned during the process of a query provide better locality information than is offered by standard *space-partitioning* ANN indices.

Let D bet a dataset of size N and dimension dim. Let I be an ANN index built from D. I provides functionality to query a vector q. We define a function $query(I, q, n)$ that searches I for neighbors of q where n is the maximum number of considered neighbors. During a search, duplicates are counted as neighbors therefore the number of unique neighbors returned may be $\leq n$. In this paper, we focus on the randomized KD-Tree algorithm as defined by Silpa-Anan and Hartley [11]. However, we believe this method can work with any *space-partitioning* style ANN index.

We start by performing a query for q in I and then calculating the similarity between q and each returned candidate neighbor. The candidates are added into a priority queue P that provides access to the candidates in ascending order by their similarity to q. After this initialization, we repeat a process of dequeueing the first candidate in P, performing a query for that candidate in I, calculating the similarity between each resultant new candidate and q, and adding these to P. The process is repeated until either P is empty or we have considered a total of n candidates including multiple encounters of the same candidate.

LAFS requires an internal query size parameter $n_s \leq n$ which dictates the number of candidates requested by each internal query. The value of n_s will vary per dataset and is chosen to produce as good of recall as possible while minimizing the amount of required similarity calculations. LAFS will perform approximately n/n_s internal queries. For example, in Figure 1 a value between $n_s = 250$ and $n_s = 1000$ would be chosen because values in this range produce the strongest recall before the curve starts its fast growth.

In order to avoid repeating queries, LAFS maintains a set S of candidates that have already been searched and does not let them be re-inserted into P. Additionally, any implementation should ensure that the similarity between q and a candidate neighbor is only calculated the first time it is encountered and not if it is found by subsequent internal queries.

Algorithm 1: Local Area Focused Search

Input: Index I, query q, similarity function σ, max number of considered neighbors n, and internal query size n_s
Output: Set of candidate nearest neighbors C

begin
  $P \leftarrow$ empty$PriorityQueue$
  $S \leftarrow \emptyset$
  $C \leftarrow query(I, q, n_s)$
  Calculate $\sigma(q, c) \quad \forall c \in C$
  $addToQueue(P, c) \quad \forall c \in C$
  $cnt = |C|$
  while $|P| > 0$ *and* $cnt < n$ **do**
    $s \leftarrow dequeue(P)$
    $S \leftarrow S \cup s$
    $C_s \leftarrow query(I, s, n_s)$
    $C_{new} \leftarrow \{c | c \in C_s, c \notin S\}$
    Calculate $\sigma(q, c) \quad \forall c \in C_{new}$
    $addToQueue(P, c) \quad \forall c \in C_{new}$
    $C \leftarrow C \cup C_{new}$
    $cnt \leftarrow cnt + |C_{new}|$
  end
  return C
end

2.3. Data Projection for General Similarity Metrics

Most *space-partitioning* ANN indexing methods require that the similarity metric be a Minkowski metric which is of the form:

$$(\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p} \tag{1}$$

Unfortunately, this type of metric is not appropriate for many types of data. To get around this limitation, dataset D can be projected using kernel methods from machine learning. The concept is that we can perform a non-linear transformation $\phi(x)$ on our data to take it from the original dim dimensional space to a much higher dimensional space where it becomes linearly separable. This projection creates a new representation of the dataset, D_ϕ, which can be compared via a Minkowski metric and, therefore, indexed by *space-partitioning* ANN methods.

A standard method of kernel projection is Kernel Principal Component Analysis (KPCA). The cost of a projecting data into the higher dimensional space is usually impractical but kernel methods, like KPCA, allow us to work only in the covariance matrix of D. The covariance between two vectors is defined as the kernel function:

$$\kappa(x, y) = \phi(x)^T \phi(y) \tag{2}$$

The kernel covariance matrix K over D is then defined as

$$K_{i,j} = \kappa(D_i, D_j) \tag{3}$$

where D_i is the i-th row of D. K is then decomposed using Principal Component Analysis (PCA) to produce a projection matrix W. As part of PCA, the dimensionality of the data can be reduced by choosing a new dimension $dimReduced < dim$ and preserving only the top $dimReduced$ eigenvectors when creating W. The projected dataset D_ϕ is then created as follows

$$D_\phi = KW \tag{4}$$

Full details about KPCA are available in [14].

KPCA becomes extremely expensive as the size of D increases due to the need to compute the full covariance of the dataset. To mitigate this problem Jiang *et al*. [7] describe a method to randomly sample $numReps$ rows from D, where $numReps << N$ and use this set to construct K. We refer to this reduced set of random representatives as R. We then build a covariance matrix Kr over R

$$Kr_{i,j} = \kappa(R_i, R_j) \tag{5}$$

PCA is applied to Kr as above to create W and may include a further dimensionality reduction by choosing a $dimReduced < numReps$. The covariance for the dataset is now computed against R

$$K_{i,j} = \kappa(D_i, R_j) \tag{6}$$

and it is projected as in equation 4.

Algorithm 2: Local Area Focused Search for Kernel Projected Data

Input: Index built from projected data I_ϕ, projected query q_ϕ, similarity function σ, max number of considered neighbors n, and internal query size n_s

Output: Set of candidate nearest neighbors C

begin
- $P \leftarrow$ empty$PriorityQueue$
- $S \leftarrow \emptyset$
- $C \leftarrow query(I_\phi, q_\phi, n_s)$
- Calculate $\sigma(raw(q_\phi), raw(c_\phi)) \quad \forall c_\phi \in C$
- $addToQueue(P, c_\phi) \quad \forall c_\phi \in C$
- $cnt = |C|$
- **while** $|P| > 0$ *and* $cnt < n$ **do**
  - $s_\phi \leftarrow dequeue(P)$
  - $S \leftarrow S \cup s_\phi$
  - $C_s \leftarrow query(I_\phi, s_\phi, n_s)$
  - $C_{new} \leftarrow \{c | c \in C_s, c \notin S\}$
  - Calculate $\sigma(raw(q_\phi), raw(c_\phi)) \quad \forall c_\phi \in C_{new}$
  - $addToQueue(P, c_\phi) \quad \forall c_\phi \in C_{new}$
  - $C \leftarrow C \cup C_{new}$
  - $cnt \leftarrow cnt + |C_{new}|$
- **end**
- **return** *C*

end

2.4. LAFS for Kernel Projected Data

LAFS is easily updated to accommodate kernel projected data. We start by using KPCA, as described in the previous section, to project D to D_ϕ and we build our index I_ϕ from D_ϕ. The projection matrix W and random representative set R are preserved and used to project query q to q_ϕ which is in the same space as D_ϕ as shown in equations 7 and 8.

$$Kr_i = \kappa(q, R_i) \tag{7}$$

$$q_\phi = KrW \tag{8}$$

An important feature of LAFS is that the similarity between a query and its candidate neighbors is calculated independently from the internal queries of I_ϕ. This means that we can use the exact similarity to prioritize candidates for subsequent internal searches instead of the approximation

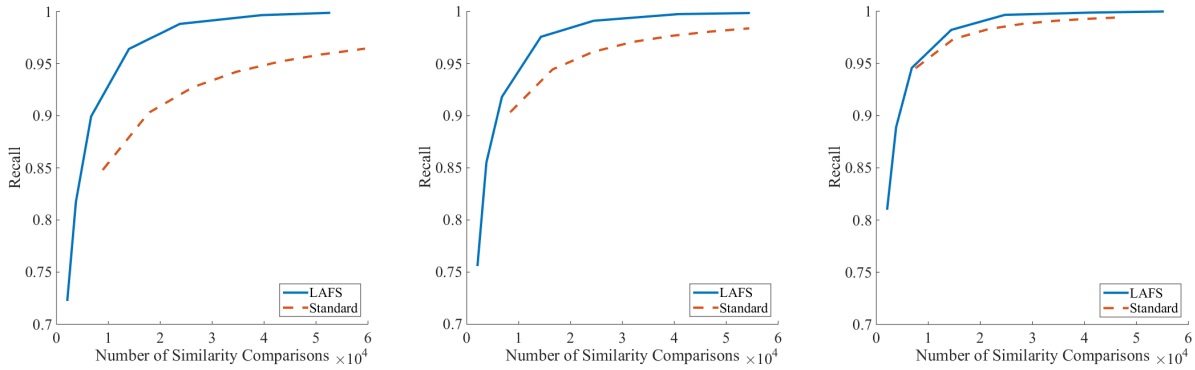

Figure 2. Performance comparison on SIFT1M for standard KD-Tree vs. KD-Tree using LAFS for indices of 5 (left), 10 (center), and 25 (right) trees.

in the projected data space. This makes it much more likely we are searching the optimal local area around the query.

The LAFS algorithm for kernel projected data, shown in Algorithm 2, is very similar to the standard algorithm. The main difference is the similarity metric is calculated on the raw (non-projected) version of the data. It is assumed that the raw version of the data can be accessed in constant time and is denoted by the $raw()$ function in the algorithm.

2.5. Complexity Discussion

Here we present a few observations on the cost of the LAFS algorithm. The complexity of a single traversal through a KD-Tree is $O(log_2(N))$ [11]. Since we are limiting our randomized KD-Tree implementation to a constant number of visited leaves (nearest neighbor candidates) n, the complexity of a query remains the same order. After candidates are generated, their similarity to the query must be calculated. While the complexity varies depending on the similarity metric, the most basic ones, such as Minkowski metrics, will be $O(dim)$. This implies that the cost of calculating the similarities will dominate algorithm time when

$$2^{dim} > N \tag{9}$$

which applies to most real-world data problems.

For each query, LAFS performs n/n_s internal queries, calculates similarities between the query and all candidates, and has additional bookkeeping costs such as managing the priority queue and set operations. Since n remains constant, none of these operations changes the overall complexity from $O(log_2(N))$ along with the $O(dim)$ for calculating the similarities. Theoretically, this is the same as using KD-Trees. In practice, depending on implementation, it is likely that LAFS will do slightly more work than a single KD-Tree query for the same number of similariy calcuations. With time being dominated by the similarity calculations, the extra work will negligible.

It is important to note that there are a number of ANN use cases where the similarity between the query and candidate nearest neighbors does not need to be calculated. In these cases, the metadata associated with the returned candidates is enough to support a decision. Since calculating similarities is central to our method, LAFS is not an appropriate choice for these use cases.

3. Performance Analysis

3.1. Analysis Setup

In this section, we will examine the effect of using LAFS over 3 different datasets. All analyses are performed using a randomized KD-Tree index. We use our own implementation which is similar to the version presented in the FLANN library [9]. For the purpose of simplicity, we will use the term *KD-Tree Index* to refer to a randomized KD-Tree index for the remainder of this section.

We compare the performance of a KD-Tree Index versus a KD-Tree Index enhanced with LAFS. As described in the previous section, the cost of calculating the similarity metrics dominates algorithm time, so we will present all results in the form of *Number of Similarity Calculations vs. Recall*. Recall is calculated as the percentage of the top k nearest neighbors returned as candidate neighbors from a query. Unless otherwise specified, $k = 10$ in the following sections.

3.2. SIFT1M

The SIFT1M dataset, provided by INRIA, is widely used for performance analysis of ANN algorithms [5]. It contains a database of 1,000,000 SIFT descriptors and a test set of 10,000 independent SIFT descriptors along with ground truth. SIFT descriptor similarity is calculated as L2 distance so we assess the performance using the basic LAFS algorithm.

Figure 2 shows the increase in performance created by

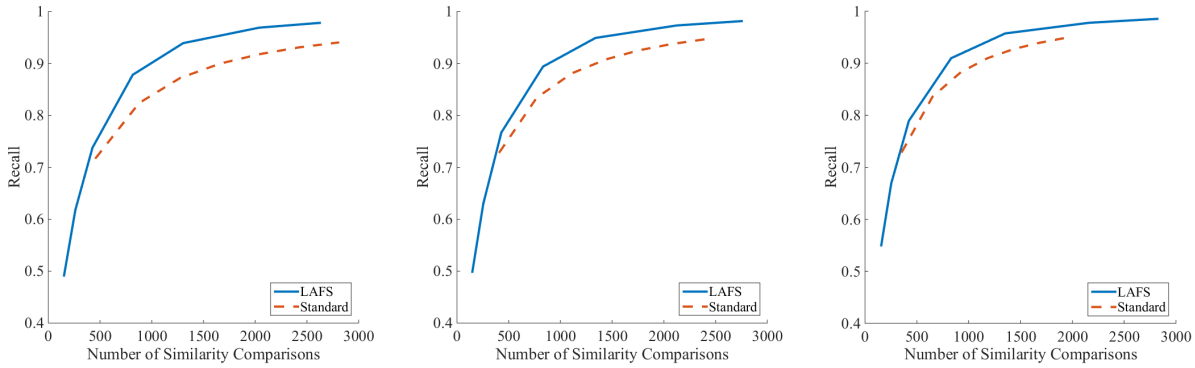

Figure 3. Performance comparison on Jittered MNIST dataset for standard KD-Tree vs. KD-Tree using LAFS for indices of 5 (left), 10 (center), and 25 (right) trees.

LAFS using $n_s = 250$. LAFS produces a significant performance increase for KD-Tree indices built with 5, 10, and 25 trees. The improvement becomes less pronounced as the number of trees increases but this is mostly likely due to the fact that the performance is approaching the upper limit in general.

Figure 5 shows the effect that the n_s parameter has on the performance of LAFS for a KD-Tree index of 10 trees. A value of $n_s = 250$ outperforms both $n_s = 50$ and $n_s = 1000$. At $n_s = 50$, we are likely searching too small a local area and not finding enough unique candidates. At $n_s = 1000$, we are likely searching too large an area and performing extra similarity comparisons without any benefit.

3.3. Jittered MNIST

Figure 4. Examples of corresponding images from MNIST (left) and Jittered MNIST (right).

The MNIST database is a widely used handwritten digits classification dataset provided by [8]. The dataset consists of 60,000 28x28 pixel images along with 10,000 independent images for testing. The images are all centered to make them directly comparable without translation.

We decided to modify MNIST in a way that allows us to test LAFS on kernel projected data using a real world image processing dataset. We introduce "jitter" into MNIST by randomly shifting each image by ± 3 pixels in both the x and y directions. See Figure 4.

The recall performance of the Jittered MNIST database is severely degraded as shown in Figure 6. For these images similarity must be measured by checking all possible alignments via 2D cross-correlation

$$xcorr_{2D}(a,b) = \arg\max_{i,j} < a, b_{i,j} > \tag{10}$$

where $b_{i,j}$ represents a shift of image b by i pixels in the x direction and j pixels in the y direction.

Cross-correlation itself is not a valid kernel but it can be converted to one via a number of methods [6]. After some experimentation, we discovered that the standard kernel for correlation

$$\kappa(a,b) = exp^{xcorr_{2D}(a,b)} \tag{11}$$

worked better than other kernels, even those proposed specifically for cross-correlation such as the one suggested in [13]. We project the data using KPCA with $numReps = 100$ and further reduce the dimensionality so that $dimReduced = 20$.

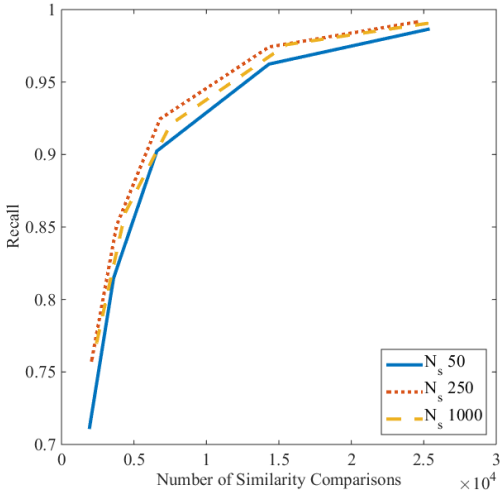

Figure 5. Effect of n_s value on recall on SIFT1M dataset using LAFS with a KD-Tree Index built with 10 trees.

Figure 3 shows a similar increase in performance as Figure 2 created by LAFS. MNIST is a relatively small dataset so we choose a smaller value for $n_s = 100$. As with SIFT1M, LAFS sees improvement for all tree configurations with its highest performance improvements on smaller numbers of trees.

3.4. Seismic Waveform Data

Seismic signals are time series recorded at seismic stations of ground motion caused by distant seismic sources such as earthquakes. One important problem in seismic signal analysis is determining the location on the Earth of the seismic event that generated an observed seismic signal. This is a very complex problem that usually requires signal detections from multiple stations and accurate models of the internal geology of the Earth.

Seismic signals have the interesting property that signals recorded at the same station from seismic events that are very nearly co-located to each other look very similar (Figure 7) but look different than signals from events that are far apart from each other. Research is ongoing into using nearest neighbor search algorithms to scan archives of historical waveforms via waveform cross-correlation searching for similar signals. Given a new signal and a similar signal from the archive, the geographic location of the event that generated the new signal can be immediately assigned to the same location that generated the historical signal [12][16]. Recent work has focused on using ANN techniques to search historical earthquake archives, which can be very large and continue to grow at a rapid rate [15].

We have developed an example dataset containing 308,219 analyst reviewed signals from known seismic events detected by the station MKAR, located in Kazakhstan, over the time period 2002-2013. We built a test set with 25,865 analyst reviewed signals from known events from the same station during 2014. These data are publicly available from the Incorporated Research Institutions for Seismology [4].

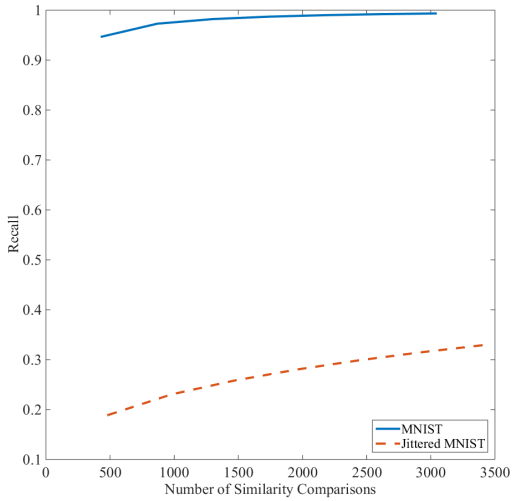

Figure 6. Performance of MNIST vs Jittered MNIST using standard KD-Tree Index built with 25 trees.

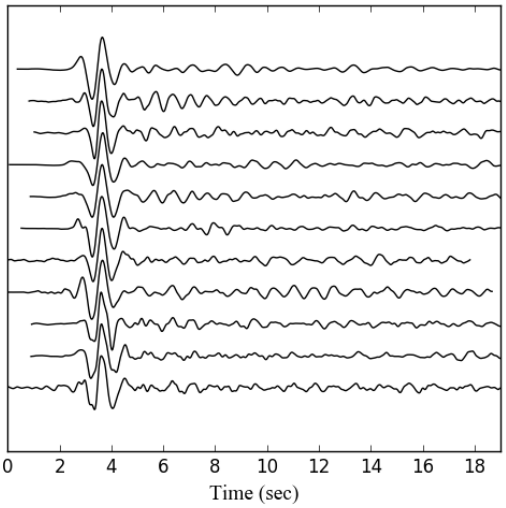

Figure 7. Example of seismic signals detected over 12 years at station MKAR in Kazakhstan from the same earthquake source in the Kuril Trench.

For each signal detection, we take a 30 second sample starting 2 seconds before the time that the signal was detected at the station. The data are sampled at 40Hz meaning that each signal is comprised of 1200 samples. The starting time for each signal is chosen by a human analyst with an estimated uncertainty of ± 0.5 seconds. Due to this uncertainty, the dataset similarity must be measured by checking a sliding set of alignments via 1D cross-correlation

$$xcorr(a,b) = \arg\max_i < a, b_i > \quad (12)$$

where b_i represents a shift of signal b by i in time. As with the jittered MNIST dataset, we use the standard correlation kernel.

$$\kappa(a,b) = exp^{xcorr(a,b)} \quad (13)$$

We project the data using KPCA with $numReps = 500$ and then further reduce the dimensionality so that $dimReduced = 200$.

Figure 8 shows a dramatic increase in performance when utilizing LAFS on seismic data with $n_s = 1000$. In seismic data, we find that outside of a few very strong matches the majority of neighbors cluster with very similar correlation scores. The kernel function is likely not discriminative enough to differentiate between these small differences. LAFS mitigates this problem by calculating the similarities in the real cross-correlation space.

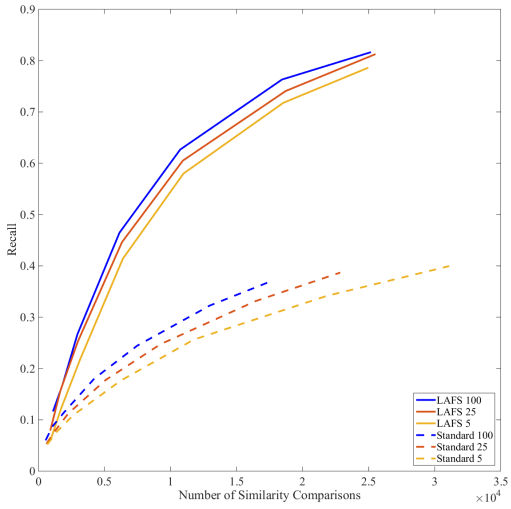

Figure 8. Performance comparison on seismic dataset for standard KD-Tree vs. KD-Tree using LAFS for indices of 5, 25, and 100 trees.

4. Conclusion

This paper presents Local Area Focused Search (LAFS). LAFS enhances standard ANN index searches by evaluating local neighborhoods over a number of small internal queries. We show that LAFS improves recall for the same amount of similarity comparisons on SIFT1M, a jittered MNIST dataset, and a seismic waveform data set. Our algorithm improves performance on kernel projected datasets by allowing similarity to be calculated in the real instead of projected space. We believe futher optimizations to LAFS are possible such as pulling candidates deeper from within the priority queue to avoid repetition and slightly expand the search space. We leave this optimization along with integration with other nearest neighbor index types such as hierarchical k-means trees or LSH as future work.

5. Acknowledgements

We would like to thank Kurt Larson, Chris Young, Sandy Ballard, Rigo Tibi, and Jonathan Woodbridge for their fantastic ideas, expertise, and technical review.

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] M. R. Abbasifard, B. Ghahremani, and H. Naderi. A survey on nearest neighbor search methods. *International Journal of Computer Applications*, 95(25):39–52, June 2014.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[3] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 577–586, New York, NY, USA, 2011. ACM.

[4] IRIS. Incorporated research institutions for seismology. `https://www.iris.edu`.

[5] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, Jan. 2011.

[6] H. Jiang and W.-K. Ching. Correlation kernels for support vector machines classification with applications in cancer data. *Computational and Mathematical Methods in Medicine*, 10.1155/2012/205025, 2012.

[7] K. Jiang, Q. Que, and B. Kulis. Revisiting kernelized locality-sensitive hashing for improved large-scale image retrieval. *CoRR*, abs/1411.4199, 2014.

[8] Y. LeCun and C. Cortes. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`, 2010.

[9] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *In VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[10] R. Paredes, E. Chávez, K. Figueroa, and G. Navarro. Practical construction of k-nearest neighbor graphs in metric spaces. In *Proceedings of the 5th International Conference on Experimental Algorithms*, WEA'06, pages 85–97, Berlin, Heidelberg, 2006. Springer-Verlag.

[11] C. Silpa-Anan and R. I. Hartley. Optimised kd-trees for fast image descriptor matching. In *CVPR*. IEEE Computer Society, 2008.

[12] M. Slinkard, S. Heck, D. Schaff, N. Bonal, D. Daily, C. Young, and P. Richards. Detection of the wenchuan aftershock sequence using waveform correlation with a composite regional network. *Bulletin of the Seismological Society of America*, 2016.

[13] G. Wachman, R. Khardon, P. Protopapas, and C. R. Alcock. Kernels for periodic time series arising in astronomy. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*, ECML PKDD '09, pages 489–505, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] Q. Wang. Kernel principal component analysis and its applications in face recognition and active shape models. *CoRR*, abs/1207.3538, 2012.

[15] C. Yoon, O. O'Reilly, K. Bergen, and G. Beroza. Earthquake detection through computationally efficient similarity search. *Science Advances*, 1(11), 2015.

[16] J. Zhang, H. Zhang, E. Chen, Y. Zheng, W. Kuang, and X. Zhang. Real-time earthquake monitoring using a search engine method. *Nature Communications*, 5:5664 doi:10.1038/ncomms664, 2014.