Towards a Learned Index Structure for Approximate Nearest Neighbor Search Query Processing

Maximilian Hünemörder, Peer Kröger, and Matthias Renz

Institute for Computer Science
Christian-Albrechts-Universität zu Kiel, Germany
{mah, pkr, mr}@informatik.uni-kiel.de

Abstract. In this short paper, we outline the idea of applying the concept of a learned index structure to approximate nearest neighbor query processing. We discuss different data partitioning approaches and show how the task of identifying the disc pages of potential hits for a given query can be solved by a predictive machine learning model. In a preliminary experimental case study we evaluate and discuss the general applicability of different partitioning approaches as well as of different predictive models.

1 Introduction

Nearest neighbor (NN) search is prevalent in many applications such as image retrieval, recommender systems, and data mining. In order to process a NN query efficiently appropriate data structures (usually called index structures) that enable identifying the result of a query by examining only a sub set of the entire data set are typically used. Additional speed-up can be gained by approximate nearest neighbor (ANN) search that trades accuracy for query time which is acceptable in many applications.

In this short paper, we examine the applicability of a new emerging paradigm, so-called learned index structures (LIS), for ANN query processing. The idea of LIS has been coined in [1] where the authors show that an index for 1D search keys (e.g. a B+-tree) is essentially similar to a regression model: the index induces an ordering of the keys and stores the data objects according to this ordering on disc pages (blocks). The corresponding learning task is, given the keys (observations) as training data, to train a predictive model (function) that determines the physical page address for each key. Processing a query is then simply applying the predictive model to the query key, i.e., predicting the addresses of the blocks (pages) on disc where the results of the query are located. While this approach works pretty well for primary key search, such as exact match queries and range queries on 1D data, we present one of the first works towards extending LIS to multi-dimensional spatial queries such as (A)NN queries.

This work aims at exploring the general applicability of LIS for multi-dimensional indexing with a focus on ANN queries. We discuss the two basic challenges any index structure has to solve (see also Figure 1). First, the database needs to be partitioned in order to store the objects in a clustered way on disc pages. We propose a new partitioning that adapts to the real data distribution and is based on a specific k-Means clustering here, but any other partitioning scheme is possible, e.g. by simply taking the

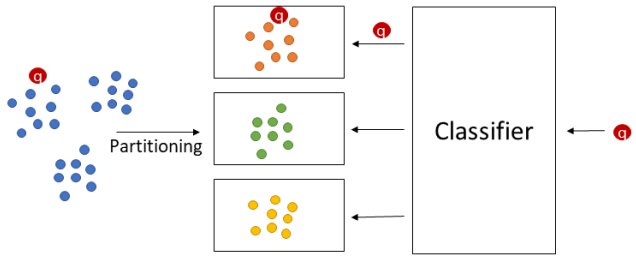

Fig. 1: A sketch of a spatial LIS: the the data (left) is partitioned and these partitions are mapped onto disc pages. A predictive model (classifier) learns this mapping. Given a query object q, the model predicts the disc page containing the potential NN of q.

leaf nodes of any hierarchical index structure. Second, the relationship between observations (values of the data objects) and their corresponding disc page IDs are learned using a predictive model. An ANN query can be supported by applying the learned prediction function to the query object. Since the predictive model may be not 100% accurate, the predicted disc page may not contain the true nearest neighbor(s) and therefore only result in an approximation. We will discuss implications, potential extensions, etc. on this aspect in detail. This way, a LIS could offer a good compromise between existing indexing paradigms: it could combine

1. a data-centric partitioning which is usually done by hierarchical index structures such as search trees that typically suffer from higher query costs due to the traversal of the search tree,
2. a fast prediction of disc page IDs which can be generally achieved by hash functions that often suffer from data-agnostic partitioning which may lead to a large number of collisions (disc page overflows) and, as a consequence to higher query times.

The reminder is organized as follows. Section 2 discusses preliminaries and related work. We sketch an LIS for multi-dimensional ANN query processing in 3. A preliminary empirical evaluation is presented in 4, and 5 offers a summary and a discussion of directions for future research.

2 Background

2.1 ANN Query Processing: Preliminaries and Related Work

Given a query q, an number $k \in \mathbb{N}$ and a distance measure $dist$, a kNN query around q on a data set $\mathcal{D}$, $\text{NN}_k(q)$, retrieves the k objects having the smallest distance to q among all objects in $\mathcal{D}$ (ties need to be resolved). Without loss of generality, we set the query parameter $k = 1$ and omit it in the following. Sequentially scanning all data objects to retrieve the NNs involves loading all pages of the entire data file from disk. Since

this is usually not acceptable performance-wise, many approaches for speeding up NN search using indexing techniques have been explored in recent years. A further way to achieve speed-ups is to trade performance for accuracy of the results using approximate algorithms that may report false hits. These ANN algorithms usually implement one of the following index paradigms:

Hierarchical indexes are typically based on balanced search trees [2–4] that recursively split the data space by some heuristics until a minimum number of objects remain in a partition. All nodes of the search tree are usually mapped to pages on disk.Searching theoretically requires $O(\log f)$ random page accesses on average for f data pages but the performance typically degrade with increasing data complexity.

Hashing such as locality sensitive hashing (LSH) and variants [5–8] applies one or more hash functions to map data objects into buckets (and store these buckets as pages on disc). If the number of objects in a bucket exceeds the maximum capacity of a page (e.g. due to an unbalanced partitioning), the objects are stored in any order on so-called "overflow pages" increasing the number of page accesses necessary to answer a given query. However, in the best case, query processing requires $O(1)$ page accesses.

Vector quantization and compression techniques (e.g. [9–11]) aim at reducing the data set size by encoding the data as a compact approximated representation such that (approximate) similarity among data objects is preserved.

A significant comparison of the different methods under varying realistic conditions is a generally challenging task. Thus, a benchmarking tool for ANN algorithms have been proposed in [12]. However, we do not aim for benchmarking LIS with other approaches here but rather explore the general applicability of LIS to ANN queries.

2.2 Learned Index Structures

The term LIS has been introduced by [1] where the authors show how to represent an index structures as a learning task. This pioneering work proposes a LIS for indexing 1D keys and supporting exact match and range queries. In recent years, the term LIS has been also used for methods that utilize machine learning techniques to support any aspect of query processing, e.g. [13] where kNN distance approximations are learned in order to support reverse NN queries, or [14] where the authors propose a new approach to generate permutations for permutation based indexing using deep neural networks. The most similar approach to ours can be found in [15] and [16] where the authors propose a learned metric index for ANN search. In contrast to our work they learn a whole tree of prediction models to index a metric space.

2.3 Contributions

LIS may offer the best of two worlds in spatial query processing, i.e., a data-centric, collision-free partitioning of the database and a search method that returns a result in constant time w.r.t. page accesses even in the worst-case. In this short paper, we explore the applicability of LIS to ANN query processing. In particular, we propose a general schema of a LIS for ANN query processing and implement this schema with existing techniques, e.g. k-means clustering for data partitioning. We present some first results

on the performance of various predictive models from machine learning and derive implications for future work.

3 Towards a Learned Index for ANN Search

The data set $\mathcal{D}$ is stored on disk in blocks (pages) of a fixed capacity c. Thus, depending on c, $\mathcal{D}$ is distributed over a set $\mathcal{P}$ of p pages on disk. Processing an object $o \in \mathcal{D}$ in RAM requires to load the entire page $P_o \in \mathcal{P}$ on which o is stored.

The key to any search index is that the data objects are not randomly distributed over $\mathcal{P}$. Rather, objects that are similar to each other w.r.t. the distance $dist$ should be placed on the same page. There are many possible solutions for producing such a clustered partitioning, e.g. using the buckets of LSH, the leaf nodes of a search tree or use an unsupervised learning method. Here, we experimented with k-means clustering, which aims at partitioning the data into k disjoint clusters maximizing the compactness of these partitions. The idea is, to use k-means in such a way, that the number of points assigned to each cluster is constrained by a minimum capacity (for efficient storage usage) c_{min} and a maximum capacity C_{max} in order to map each cluster to one data page (C_{max} usually depends on c from above). Extensions such as Constraint k-means [17] are able to cope with these issues but are computationally very complex. Instead, in our study, we propose to just use traditional k-means clustering. The points assigned to a cluster $C_i (1 \leq i \leq k)$ are mapped to page $P_i \in \mathcal{P}$.

For query processing, we need to predict the page $P \in \mathcal{P}$, the query object q would have been placed on. This page likely contains the NN of q (depending on the partitioning, etc.). This prediction could be done by any machine learning model that can learn the mapping of an object to the corresponding disk page. Analogously to hashing, such a predictive model is a function

$$M : \mathbb{F} \rightarrow \mathcal{P}$$

from the feature space $\mathbb{F}$ of the data into the set of data pages that depends on some model-specific parameters θ_M. In general, we can learn (train) the corresponding parameters θ_M from $\mathcal{D}$ (and the corresponding partitioning $C_1, ..., C_k$). Given a query object $q \in \mathbb{F}$ and a predictive model M trained on $\mathcal{D}$, we can predict the disk page $P = M(q)$ by applying M on q. The page P can be loaded into main memory and the NN of q among all objects stored on P can be determined and returned as (approximate) result. Since our data partitioning does not produce overflow pages, we only need to access one page, i.e., $P = M(q)$. Thus, the time complexity is guaranteed to be in $O(1)$ in any cases (we can usually even assume that the model M fits into main memory). The accuracy of this procedure obviously depends on various aspects such as the accuracy of the prediction, the data partitioning, etc., some will be examined in Section 4. However, we consider the optimization of such aspects as an open challenge for future research, e.g. by aggregating more information from the partitions such as centrality measures, distance bounds, etc.

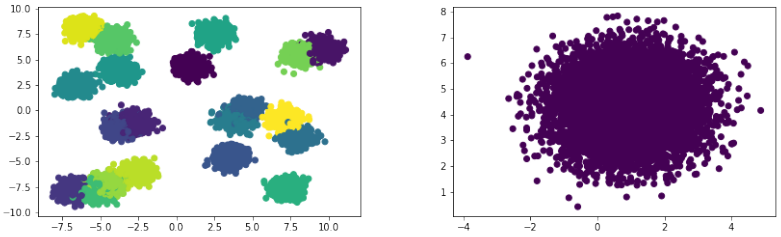

Fig. 2: Random 2D projections of sample clustered (left) and non-clustered (right) data.

4 Evaluation

4.1 Set-up

In order to get a first impression of the proposed LIS for ANN query processing, we used synthetic data sets generated by the *make_blobs* function from sklearn[1]. In all experiments, we generated five different random datasets and report average results. We conducted two general runs w.r.t. the data distributions: clustered and non-clustered data. Figure 2 depicts arbitrary 2D projections of two sample data sets from both runs. We used 20-dimensional synthetic datasets consisting of 5000, 10000, 30000 and 50000 samples. The clustered datasets had 20 clusters with a cluster standart deviation of 0.5 and the non-clustered datasets have only a single gaussian blob with a standard deviation of 1.0. Additionally, we used a low dimensional embedding of the popular MNIST data set generated by a fully connected Autoencoder (AE). Since this paper is a preliminary study of the general applicability of LIS to ANN search we did not yet compare to other ANN methods.

We used two different accuracy scores for evaluation. First, to explore the potential of the different predictive models to learn the mapping of objects to pages, we employed a classical train-validation split (called **validation accuracy**). Second, to measure the approximation accuracy of the query (called **test accuracy**), we used a withheld third sub-set of the data (not used in partitioning or training of the predictive model) as query objects, compared the results of these queries with the correct NN computed by a brute force search. The accuracy is determined by the ratio of the amount of zero distance hits and the amount of query objects. Additionally, we report the mean relative error for ANN search in our repository[2]).

For the partitioning step, we used the k-means implementation from sklearn. For comparison, we used the leaf nodes of a kd-tree (also from sklearn) as an alternative data partitioning. As predictive models, we used diverse classifiers from sklearn, including: Naïve Bayes, Decision Tree and Random Forest, Support Vector Machine (SVM) with a linear and an rbf kernel, and a simple dense multi-layer perceptron (MLP). For

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html`

[2] `https://github.com/huenemoerder/kmean-lis.git`

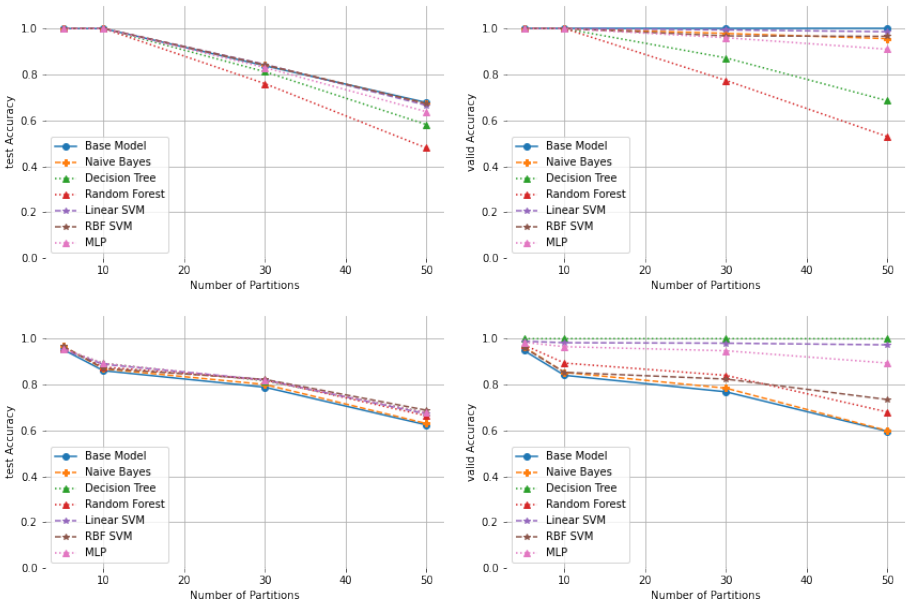

Fig. 3: Test accuracy (left charts) and validation accuracy (right charts) on **clustered** data sets (upper charts: k-means partitioning; lower charts: kdtree partitioning).

these preliminary experiments we did not perform hyper-parameter tuning but used reasonable default parameters. As a "Base Model", we assign each query object to its closest centroid of the corresponding partition (validation accuracy of 1.0 by design). The "size" of this model grows linearly with the number of partitions, i.e., database size, and is expected to not fit into the cache (requiring additional page accesses on application). The AE for the MNIST data set was implemented in pytorch[3] with only one single linear layer that maps the flattened images (784 dimensional array) to a latent space vector of 32 dimensions (usin Leaky ReLU as activation).

4.2 Results

We analysed the relationship between the test accuracy and the number of samples and number of partitions, i.e., data pages. In all runs, we kept the capacity of pages fixed but changed the number of data points n accordingly. Figure 3 displays this relationship on clustered data sets. In general, we can see that both the test accuracy and the validation accuracy drops with increasing number of partitions. This is somehow intuitive: with increasing number of partitions (and data points), the mapping that has to be learned by the predictive model becomes more and more complex. It is interesting to note that for most models the validation error (right charts) remains better than the test accuracy (left charts), i.e. even though, the mapping is learned well, the true NNs for the query

[3] `https://pytorch.org/`

objects are approximated not quite as well. In these cases, the partitioning model seems to not optimally fit the real data distribution and therefore even with a perfect predictive model some queries can be placed in an unsuitable data page. This is also reflected in the fact that the kd-tree partitioning performs even worse in terms of test accuracy, since the clustered dataset was created in a way that favours k-means. We can also observe that the Decision Tree classifier shows perfect validation accuracy for the kd-tree partitioning, while showing the worst performance for k-means. This suggests that choosing a fitting pair of prediction and partitioning algorithm is vital to at least result in a high validation accuracy. These observations are further confirmed by the non-clustered data sets (the results can be found in our repository[4]). Additionally, this is further reflected in our results on MNIST in Table 1, where the test accuracies for the kdtree paritioning are significantly worse than the ones for k-means. Generally further experiments and benchmarking are obviously necessary to obtain more significant results.

5 Summary

In this short paper, we applied the idea of LIS to ANN query processing and examined its general applicability to this problem. We explored a new data partitioning based on k-means clustering and applied the standard predictive models from machine learning in a simple set up. The results are generally promising for synthetic (clustered/non-clustered) and real data such that we think it is worth putting more future focus on LIS. For example, exploring new ways for data partitioning including a more thorough evaluation of different existing partitioning schemes could be interesting. Also, understanding the relationship between data characteristics, properties of the partitioning, and the accuracy of different predictive models could be a promising research direction that may lead to approaches that better integrate partitioning and learning. Additionally, exploring postprocessing methods to increase accuracy, e.g. use additional information from training as well as from the partitioning like distance bounds would be helpful. Last not least, the application of LIS to other types of similarity queries is still an open research question.

[4] `https://github.com/huenemoerder/kmean-lis.git`

Table 1: Results on MNIST data set ($k$menas partitioning)

	k-means		KDTree	
Classifier	Validation Accuracy	Test Accuracy	Validation Accuracy	Test Accuracy
Base Model	**1.000**	0.8808	0.5407	0.4974
Naïve Bayes	0.9140	0.8479	0.6140	0.5409
Decision Tree	0.8560	0.8121	**0.9997**	0.6160
Random Forest	0.7315	0.7089	0.4610	0.4066
Linear SVM	0.9973	0.8800	0.9630	0.6165
RBF SVM	0.9845	**0.8810**	0.8588	**0.6388**
MLP	0.9455	0.8736	0.8678	0.5994

References

1. Kraska, T., Beutel, A., Chi, E.H., Dean, J., Polyzotis, N.: The case for learned index structures. In: Proc. Int. Conf. on Management of Data (SIGMOD), Houston, TX. (2018) 489–504
2. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Proc. Int. Conf. on Very Large Databases (VLDB). (1997)
3. Sakurai, Y., Yoshikawa, M., Uemura, S., Kojima, H., et al.: The a-tree: An index structure for high-dimensional spaces using relative approximation. In: Proc. Int. Conf. on Very Large Databases (VLDB). (2000) 5–16
4. Amsaleg, L., Jónsson, B.Þ., Lejsek, H.: Scalability of the nv-tree: Three experiments. In: Proc. Int. Conf. on Similarity Search and Applications (SISAP), Lima, Peru. Volume 11223 of LNCS., Springer (2018) 59–72
5. Christiani, T.: Fast locality-sensitive hashing frameworks for approximate near neighbor search. In: Proc. Int. Conf. on Similarity Search and Applications (SISAP), Newark, NJ. Volume 11807 of LNCS., Springer (2019) 3–17
6. Jafari, O., Nagarkar, P., Montaño, J.: mmlsh: A practical and efficient technique for processing approximate nearest neighbor queries on multimedia data. In: Proc. Int. Conf. on Similarity Search and Applications (SISAP), Copenhagen, Denmark. Volume 12440 of LNCS., Springer (2020) 47–61
7. Jafari, O., Nagarkar, P., Montaño, J.: Improving locality sensitive hashing by efficiently finding projected nearest neighbors. In: Proc. Int. Conf. on Similarity Search and Applications (SISAP), Copenhagen, Denmark. Volume 12440 of LNCS., Springer (2020) 323–337
8. Ahle, T.D.: On the problem of p_{1-1} in locality-sensitive hashing. In: Proc. Int. Conf. on Similarity Search and Applications (SISAP), Copenhagen, Denmark. Volume 12440 of LNCS., Springer (2020) 85–93
9. Weber, R., Schek, H.J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proc. Int. Conf. on Very Large Databases (VLDB). (1998) 194–205
10. Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., Abbadi, A.E.: Vector approximation based indexing for non-uniform high dimensional data sets. In: Proc ACM Int. Conf. on Information and Knowledge Management (CIKM), McLean, VA. (2000) 202–209
11. Houle, M.E., Oria, V., Rohloff, K., Wali, A.M.: Lid-fingerprint: A local intrinsic dimensionality-based fingerprinting method. In: Proc. Int. Conf. on Similarity Search and Applications (SISAP), Lima, Peru. Volume 11223 of LNCS., Springer (2018) 134–147
12. Aumüller, M., Bernhardsson, E., Faithfull, A.J.: Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In: Proc. Int. Conf. on Similarity Search and Applications (SISAP), Munich, Germany. Volume 10609 of LNCS., Springer (2017) 34–49
13. Berrendorf, M., Borutta, F., Kröger, P.: k-distance approximation for memory-efficient rknn retrieval. In: Proc. Int. Conf. on Similarity Search and Applications (SISAP), Newark, NJ. Volume 11807 of LNCS., Springer (2019) 57–71
14. Amato, G., Falchi, F., Gennaro, C., Vadicamo, L.: Deep permutations: Deep convolutional neural networks and permutation-based indexing. In: Proc. Int. Conf. on Similarity Search and Applications (SISAP), Tokyo, Japan. Volume 9939 of LNCS. (2016) 93–106
15. Antol, M., Ol'ha, J., Slanináková, T., Dohnal, V.: Learned metric index—proposition of learned indexing for unstructured data. Information Systems **100** (2021) 101774
16. Slanináková, T., Antol, M., Ol'ha, J., Vojtěch, K., Dohnal, V.: Data-driven learned metric index: an unsupervised approach. In: International Conference on Similarity Search and Applications, Springer (2021) To appear.
17. Bennett, K., Bradley, P., Demiriz, A.: Constrained k-means clustering. In: Technical Report MSR-TR-2000-65, Microsoft Research. (2000)