Learned k-NN Distance Estimation

Daichi Amagata
Osaka University
Japan
amagata.daichi@ist.osaka-u.ac.jp

Yusuke Arai
Osaka University
Japan
arai.yusuke@ist.osaka-u.ac.jp

Sumio Fujita
Yahoo Japan Corporation
Japan
sufujita@yahoo-corp.jp

Takahiro Hara
Osaka University
Japan
hara@ist.osaka-u.ac.jp

ABSTRACT
Big data mining is well known to be an important task for data science, because it can provide useful observations and new knowledge hidden in given large datasets. Proximity-based data analysis is particularly utilized in many real-life applications. In such analysis, the distances to k nearest neighbors are usually employed, thus its main bottleneck is derived from data retrieval. Much efforts have been made to improve the efficiency of these analyses. However, they still incur large costs, because they essentially need many data accesses. To avoid this issue, we propose a machine-learning technique that quickly and accurately estimates the k-NN distances (i.e., distances to the k nearest neighbors) of a given query. We train a fully connected neural network model and utilize pivots to achieve accurate estimation. Our model is designed to have useful advantages: it infers distances to the k-NNs at a time, its inference time is $O(1)$ (no data accesses are incurred), but it keeps high accuracy. Our experimental results and case studies on real datasets demonstrate the efficiency and effectiveness of our solution.

1 INTRODUCTION
Due to the proliferation of IoT devices, it is becoming easier to maintain big data. Big data mining is well known to be an important task for data science, because it can provide useful observations and new knowledge hidden in given large datasets. Proximity-based data analysis is particularly utilized in many real-life applications [11, 34].

Motivation. Many proximity-based analysis tools employ distances to k nearest neighbors (k-NNs), because they are an intuitive and effective way to measure the density (or sparsity) of a given point. For example, geometric inference [10], outlier/anomaly detection [3, 4], clustering [13], and cell analysis [58] employ this approach. Below, by using some representative analysis tools that are used by industrial applications, we introduce how to use the distances to k-NNs more concretely.

Example 1 (k-NN density estimation & visualization). Density estimation [32] is one of the data visualization tools used most frequently, e.g., for detecting hot spots w.r.t. traffic and crime. Given a query point $q$, its density based on k-NNs is obtained via the following estimator [10]:

$$\hat{\rho}(q) = \frac{1}{nV_d} \left( \frac{k}{\sum_{j=1}^{k} d_{q_j}/2} \right)^{d/2},$$

(1)

where $n$ is the dataset size, $V_d$ is the volume of the unit hypersphere in a d-dimensional Euclidean space, $x_q^j$ is the j-th NN of $q$, and $\text{dist}(\cdot, \cdot)$ is the Euclidean distance between two points. Given a space for visualization and its resolution, each pixel can be a query $q$ and obtains its k-NN density via the above equation, which is then displayed in the corresponding (sub-)space. (The visualized result of a crime position dataset may be used by security service companies/organizations to highlight danger areas.)

Example 2 (Distance-based outlier detection). The importance of outlier detection and its wide range of applications are well known, e.g., see survey papers [11, 17, 48]. Let us consider machine learning (ML) models, and they are now used to automate operations in many companies. However, ML models suffer from data errors [43], and such errors (outliers) should be detected and removed to train high performance models. Distance-based outlier detection is often utilized to achieve this, because it is intuitive and effective [3]. Literature [27] proposed a definition that a point $x$ is an outlier iff the distance between $x$ and its k-NN is larger than a threshold. Similarly, literature [40] proposed a definition that a point $x$ is an outlier iff the distance between $x$ and its k-NN is in the top-N among a given dataset.

Example 3 (Reverse engineering for clustering). Let us consider density-peaks clustering (DPC) [42], one of the density-based clustering algorithms. DPC has been employed in many domains [2]. For example, [55] used DPC to analyze industrial IoT datasets. As a criterion for clustering, DPC computes the local density $\rho_i$ for each point $x_i$, and it is the number of points $x_j$ such that $\text{dist}(x_i, x_j) \leq d_{\text{cut}}$, where $d_{\text{cut}}$ is a threshold. If $\rho_i > \rho_{\text{min}}$, DPC regards $x_i$ as noise, similarly to the distance-based outlier.

Assume that a result of DPC is given but the value of $d_{\text{cut}}$ is unknown. Assume furthermore that we know the number of noises (from clustering labels) and $\rho_{\text{min}}$. To reproduce the clustering result, we need to know $d_{\text{cut}}$, but, because $d_{\text{cut}}$ is a real value, testing all possible values is not practical. Then, it is important to notice that, by measuring the distance to $\rho_{\text{min}}$-NN, we can see the distance that distinguishes noises. (Details will be given in Section 5.3.) From this observation, we can estimate $d_{\text{cut}}$.

The above examples clarify the importance of obtaining the distances to k-NNs.

*Both authors contributed equally to this research.*
A straightforward approach to computing the \( k \)-NN distances is to retrieve the \( k \)-NN points. Because the above applications face large datasets, there are many approaches that efficiently retrieve the \( k \)-NN points, such as tree indices \([8, 9, 39]\). However, when the objective is to know the distances to \( k \)-NNs, data retrieval approaches still incur large computational cost, because they need to access many points.

**Contribution.** In this paper, we consider a machine-learning (ML) technique to alleviate this cost. Recently, ML techniques have been introduced in many database management systems \([21, 34, 44, 54]\) because of their adaptability to the distributions of real datasets and accurate inference ability. It is important to note that, to employ an ML technique for the problem of \( k \)-NN distance estimation, the ML model has to be simple (to enable fast inference) yet accurate. We tackle this challenge with a regression-based solution.

Assume that we have a set \( X \) of \( d \)-dimensional points, there are a lot of pivot points in the data space, and the distances to the \( k \)-NNs of the pivots are pre-computed. Given a query point \( q \), there would exist a pivot that is very near \( q \). Therefore, by using the \( k \)-NN distances of the pivot, we can infer the distances to the \( k \)-NNs of \( q \) and its error would be very small. It is however not feasible, in practice, to prepare pivots so that there exist sufficiently close pivots for arbitrary queries, as this approach is prohibitive w.r.t. space cost. Hence, we train a regression model based on a fully connected neural network and exploit a reasonable number of pivots to learn an accurate model. Our model, namely PivNet, is designed to deal with multiple values of \( k \) above application examples, our \( k \)-NN distance estimation enables to obtain a threshold for \( k \)-NN search. This speeds-up the \( k \)-NN query processing because we do not need to set \( \infty \) as the initial threshold any more (although the result may be approximate). To summarize, our main contributions are as follows:

- We propose PivNet, a learned model that infers the \( k \)-NN distances of a given query in \( O(1) \) time. Its source code is open-source at https://github.com/arally/pivnet.
- We conduct experiments to demonstrate that PivNet provides a fast inference time while keeping small errors.
- We conduct case studies w.r.t. \( k \)-NN density visualization, distance-based outlier detection, reverse engineering for density-based clustering, and approximate \( k \)-NN search. The results show the practical efficiency and effectiveness of PivNet.

This is a full version of \([1]\).

**Organization.** The remainder of this paper is organized as follows. Section 2 formally defines our problem. Section 3 presents our solution. Sections 4 and 5 respectively report our experimental and case studies. Finally, Section 7 concludes this paper.

## 2 PRELIMINARY

Let \( X \) be a set of \( d \)-dimensional points in the Euclidean space. We use \( \text{dist}(x, x') \) to denote the Euclidean distance between two points \( x \) and \( x' \). Below, we define the \( k \) nearest neighbors (\( k \)-NNs).

**Definition 1** (\( k \) nearest neighbors). Given a set \( X \) of points, a query point \( q \), and \( k \), \( A \) is a set of the \( k \) nearest neighbors of \( q \) that satisfies \(|A| = k \) and \( \forall x \in A, \forall x' \in X \setminus A, \text{dist}(x, q) \leq \text{dist}(x', q) \).

Then, our problem is defined as follows:

**Definition 2** (\( k \)-NN distance estimation problem). Given a set \( X \) of points, a query \( q \), and \( k \), let \( A \) be a set of the \( k \) nearest neighbors of \( q \) defined in Definition 1. This problem estimates the distance between \( q \) and \( x \) for each \( x \in A \).

We assume that \( X \) is static (or is not frequently updated), which is a common assumption, e.g., for applications in Examples 1–3. Furthermore, we assume that there is a maximum \( k \), denoted by \( k_{\text{max}} \) (e.g., \( k_{\text{max}} \) is 50 or 100), which is specified by applications (so that it satisfies their requirements). Our solution is then designed to estimate the distances to \( k_{\text{max}} \)-NNs.

Consider a query \( q \) and let \( \hat{\text{dist}}(q, x_p^k) \) be an estimated distance to its \( k \)-NN. For distance estimation, minimizing absolute error \( |\text{dist}(q, x_p^k) - \text{dist}(q, x_q^k)| \) is clearly important, as can be seen from Examples 2 and 3. Therefore, as an error metric, we mainly use MAE (Mean Absolute Error), and we compute the MAE of \( q \), \( \text{MAE}_q \), as follows:

\[
\text{MAE}_q = \frac{\sum_{k=1}^{k_{\text{max}}} |\text{dist}(q, x_p^k) - \text{dist}(q, x_q^k)|}{k_{\text{max}}}.
\]

(In our empirical evaluation, Mean Absolute Percentage Error, MAPE, is also measured.)

Because the application examples in Section 1 typically assume low-dimensional datasets \([2, 14–16, 50]\), this paper considers that \( d \) is low. The case of high-dimensional data is a future work, because solutions for this case can be totally different\(^1\).

## 3 OUR SOLUTION

**Main idea.** The main requirements of the \( k \)-NN distance estimation problem are high efficiency and accuracy. To satisfy these requirements, our first idea uses pivot points. Consider that a lot of pivots are distributed in the data space \( \mathbb{R}^d \). Specifically, the data space is divided equally by fine-grained grid cells and each cell has a pivot \( p \) centered at the cell. Given a query \( q \), its corresponding cell can be found in \( O(1) \) time. Let \( x_p^i \) be the \( i \)-th NN of \( p \) in \( X \). From triangle inequality, if \( x_p^i \notin \{x_q^k|i \in [1, k - 1]\} \), we have

\[
\text{dist}(q, x_p^k) \leq \text{dist}(q, p) + \text{dist}(p, x_p^k).
\]

Besides, in a very fine-grained grid, we can have \( \text{dist}(q, p) = 0 \). This suggests that the distances to the \( k \)NNs of \( q \) can be obtained from those of \( p \). Note that \( \text{dist}(p, x_p^i) \), where \( i \in [1, k_{\text{max}}] \), can be obtained efficiently in a pre-processing phase (by using a tree index, e.g., \( k \)-d-tree). That is, for \( k_{\text{max}} = O(1) \), this approach infers the \( k \)NN distances of \( q \) in \( O(1) \) time and its accuracy would be high.

Although the above idea functions well in theory, preparing such an unlimited number of pivots is space-consuming and is not feasible in practice. Now the challenge is how to achieve the above high accuracy while keeping \( O(1) \) inference time and the usefulness of pivots. We overcome this challenge by using a neural network model, because it can effectively learn the distributions of

---

\(^1\)This is trivial. For example, space partitioning methods, such as tree indices, are generally used for similarity search on low-dimensional data, whereas they are not used for high-dimensional data \([50]\).
Figure 1: The left part shows the data space and a grid. Blue stars centered at cells represent pivots and the red star represents a query \( q \). The right part shows our five-layer fully connected neural network model. The input layer has the coordinates of \( q, \text{dist}(q,p_2) \), where \( p_2 \) is the nearest pivot of \( q \), and \( \text{dist}(p_2,x_k^q) \) \((k \in [1,k_{max}])\). It has three hidden layers, and its output layer is regarded as a vector \( \langle \text{dist}(q,x_1^q), \ldots, \text{dist}(q,x_{k_{max}}^q) \rangle \).

real datasets. We incorporate the idea that the kNN distances of a pivot near \( q \) can be similar to those of \( q \) into the neural network, to train an accurate estimation model. This is our second idea.

**Modeling by a neural network.** First, we divide each coordinate of the data space equally to build a grid offline. The granularity is application-dependent and can be specified based on memory size. Each cell of the grid has a pivot \( p \) at its centroid. Note that, given a query \( q \), its nearest pivot is obtained in \( O(1) \) time because pivots are centroids of their cells.

Then, to implement our main ideas, we consider the kNN distance estimation problem as a regression problem that outputs \( \text{dist}(q,x_k^q) \). By preparing a set \( Q_{train} \) of training queries, we can obtain \( \text{dist}(q,x_k^q) \) offline. Therefore, we can run supervised learning to train the regression model. We employ a neural network to achieve this, motivated by the fact that it provides high inference accuracy in many problems, domains, and fields. It is important to recall that the inference time has to be fast while minimizing estimation errors, as mentioned in Section 1. To achieve this, a simple yet effective model is required. (Neural networks with complex and/or many layers cannot achieve this, because they increase the number of running costly matrix multiplication.) We specifically utilize a five-layer (three hidden layers) fully connected neural network and use ReLU \([35]\) as an activation function\(^2\). As the input of this neural network, we use \( q, k, \text{dist}(q,p) \), where \( p \) is the nearest pivot of \( q \), and \( \text{dist}(p,x_k^p) \). Via stochastic gradient descent, this neural network is trained to minimize the following loss function:

\[
\mathcal{L} = \frac{1}{k_{max}|Q_{train}|} \sum_{(q,k) \in P} |\text{dist}(q,x_k^q) - \hat{\text{dist}}(q,x_k^q)|,
\]

where \( P \) is a set of all pairs of \( q \in Q_{train} \) and \( k \in [1,k_{max}] \), and \( \hat{\text{dist}}(q,x_k^q) \) is obtained by the neural network \( f \), i.e.,

\[
\hat{\text{dist}}(q,x_k^q) = f(q,k,\text{dist}(q,p),\text{dist}(p,x_k^p)).
\]

\(^2\)Our experimental result shows that a deeper neural network does not yield a clear advantage.

By adding \( \text{dist}(q,p) \) and \( \text{dist}(p,x_k^p) \) as feature values, its inference accuracy becomes higher than that of its variant without them while keeping simpleness, see Section 4.

**Optimization.** The above neural network has several drawbacks. First, it needs to deal with each \( k \in [1,k_{max}] \) iteratively, as it outputs only \( \text{dist}(q,x_k^q) \). As described in Definition 2, when we need distances to \( k \)-NNs, this approach incurs \( O(k) \) time. For example, in Example 1, \( k \) is not small, so this cost is not negligible. Moreover, this approach incurs a long training time because we need to consider each pair of \( q \in Q_{train} \) and \( k \in [1,k_{max}] \).

To remove these drawbacks, we optimize our design of the neural network and propose PivNet, which is illustrated in Figure 1. The last layer of this neural network has \( k_{max} \) neurons, and this model is trained so that it can infer the distances to \( k_{max} \)-NNs at a time. This model, \( \text{frtioNet} \), requires \( q,k,\text{dist}(q,p) \), where \( p \) is the nearest pivot of \( q \), and \( \langle \text{dist}(p,x_1^p),\ldots,\text{dist}(p,x_{k_{max}}^p) \rangle \) as input, and it outputs estimated \( k_{max} \)-NN distances. That is,

\[
\hat{v}_q = \text{frtioNet}(q,\text{dist}(q,p),v_p),\quad v_p = \langle \text{dist}(p,x_1^p),\ldots,\text{dist}(p,x_{k_{max}}^p) \rangle,
\]

\[
v_q = \langle \text{dist}(q,x_1^q),\ldots,\text{dist}(q,x_{k_{max}}^q) \rangle.
\]

The loss function of PivNet is also optimized to deal with the distances to \( k_{max} \)-NNs at the same time while achieving accurate regression. Let \( || \cdot ||_1 \) be the \( L_1 \) norm of a given vector. To obtain a similar accuracy to that of \( f \), PivNet specifically utilizes the following loss function:

\[
\mathcal{L}_{opt} = \frac{1}{|Q_{train}|} \sum_{q \in Q_{train}} ||v_q - \hat{v}_q||_1 / k_{max}.
\]

(Notice that \( v_q \) is the ground truth.) We train PivNet so that \( \mathcal{L}_{opt} \) is minimized. Then, as long as \( k \leq k_{max} \), PivNet infers the distances to the \( k \)-NNs of \( q \) in \( O(1) \) time (for fixed hyper-parameters) without losing accurate estimation ability.

**Preparing query training set.** A straightforward approach to preparing queries for training is to randomly sample points from \( X \). In this case, however, the model overfits the distribution of \( X \). When
queries are specified uniformly at random in $\mathbb{R}^d$ as in Example 1, the performance of this model would degrade, particularly when queries are specified at sparse spaces. To avoid this drawback, we augment training queries by generating synthetic ones that follow a uniform distribution in $\mathbb{R}^d$ (like pivots), and they are added into $Q_{train}$. After that, for each $q \in Q_{train}$, we compute its $k$-NNs among $X \setminus Q_{train}$ and train PivNet.

### 4 EXPERIMENT

This section presents our experimental results. All experiments were conducted on a Ubuntu 18.04 LTS machine with 3.0GHz Core i9-10980XE CPU and 128GB RAM.

#### 4.1 Setting

**Methods.** We compared PivNet with the following approaches:

- $kd$-tree [8]: This returns the exact answer by retrieving the $k$-NNs via a branch-and-bound algorithm.
- LightGBM [26]: This is a state-of-the-art gradient boosting decision tree-based regression model. For its input, we used the same feature values as those of PivNet, to be fair.
- Pivot: This algorithm uses the right side of Equation (3) as inference of the $k$-NN distances.
- QueryNet: This is a variant of PivNet and does not utilize pivots. This model also estimates the distances to $k_{\max}$-NNs at a time.
- PivNet-itr: This is a variant of PivNet, and it infers the distance to $k$-NN for each $k \in [1, k_{\max}]$ iteratively.

The above methods were implemented by Python, were single threaded, and ran in-memory. (We have no other competitors, because our problem has no prior works.) QueryNet, PivNet-itr, and PivNet were trained in PyTorch. For $kd$-tree, we used SciPy.

**Datasets.** We used the following publicly available real datasets for evaluation. In them, we removed points with missing values.

- **Crime** [15]: A set of 259,809 geo-location (i.e., 2D) points where crimes occurred in Atlanta, U.S.A.
- **HEPMASS** [6]: A set of 10,049,359 5D mass feature datasets.
- **Household** [2]: A set of 1,771,612 power consumption data measured in a house. We used 4 attributes (active power, reactive power, voltage, and intensity).
- **PAMAP2** [41]: A set of 2,643,873 5D-sensor data.
- **Places** [5]: A set of 9,033,486 geo-locations of public places in the U.S.A.
- **Wisdom** [53]: A set of 4,476,481 3D gyroscopes sensors obtained from smartphones and smartwatches.

For each dataset, we randomly choose 100,000 points in the dataset and added them into $Q_{train}$. Moreover, we added 100,000 points augmented by the approach presented in Section 3 into $Q_{train}$. We used random 80\% points in $Q_{train}$ for training and the remaining points for validation. We randomly choose another 10,000 points in the dataset, augmented 10,000 points in the above manner, and used them as test queries. A set of the remaining points in the dataset was used as $X$.

**Hyper-parameters.** We set $k_{\max} = 50$. For the grid introduced in Section 3, we defined a constant $c$ to equally divide the coordinate of each domain. For 2D and 3D datasets, $c = 2048$ and $c = 256$, respectively. For 4D and 5D datasets, $c = 32$.

The numbers of neurons in the second and third layers were 64 on Crime and Places and 128 on the others. The number of neurons in the fourth layer was 32 on all datasets. The batch size was 500. The learning rate of QueryNet was 0.1, 0.1, 0.1, 0.2, 0.2, and 0.1 on Crime, HEPMASS, Household, PAMAP2, Places, and Wisdom, respectively. Also, that of PivNet-itr was 0.03 (0.01), 0.02 (0.003), 0.01 (0.005), 0.04 (0.003), 0.01 (0.003), and 0.02 (0.005) on Crime, HEPMASS, Household, PAMAP2, Places, and Wisdom, respectively.

**Evaluation criteria.** To measure the accuracy of each inference model, we computed the average and median of $MAE_q$, which is defined in Equation (2), and those of $MAPE_q$ among all queries in the test query set. Note that $MAPE_q$ is obtained as follows:

$$MAPE_q = \frac{1}{k_{\max}} \sum_{k=1}^{k_{\max}} \frac{|dist(q, x^k_q) - \hat{dist}(q, x^\hat{k}_q)|}{\hat{dist}(q, x^\hat{k}_q)}.$$ 

As for running time, we measured the average time to obtain the $k_{\max}$-NN distances of a given query. In addition, we measured the training time of each ML model.

#### 4.2 Result

Tables 1 and 2 respectively show the results of MAE and MAPE (smaller is better), while Table 3 shows the result of time.

**Effectiveness of using neural network.** First, let us compare PivNet (or PivNet-itr) with Pivot to study the effectiveness of utilizing neural network. Tables 1 and 2 show that the accuracy of PivNet-itr is better than that of Pivot. This result suggests that our neural network does not simply use the $k$-NN distances of pivots as inference but learns an accurate regression model. Notice that Pivot provides upper-bounds of $k$-NN distances, see Equation (3). These bounds become loose as $d$ increases. For example, Pivot yields a small MAE on Crime and Places ($d = 2$) but its accuracy degrades on PAMAP2 ($d = 5$).

Moreover, we see that PivNet yields better accuracy than LightGBM in most tests. This result also justifies our choice of utilizing a neural network.

**Effectiveness of using pivots.** We next investigate the usefulness of pivots by comparing PivNet with QueryNet. From Table 1, we see that PivNet is always better than QueryNet. Also, Table 2 shows that PivNet is better than QueryNet with only a single exception of the average case of PAMAP2. (Recall that, as mentioned in Section 2, we weight MAE.) For example, PivNet provides about 3.9x (3.7x), 1.1x (1.4x), 1.9x (1.8x), 1.5x (1.3x), 6.9x (7.3x), and 4.1x (3.3x) less MAE than QueryNet in the average (median) case on Crime, HEPMASS, Household, PAMAP2, Places, and Wisdom, respectively.

**Comparison with PivNet-itr.** For PivNet and PivNet-itr, we study how the difference in neural network structures impacts their performances. Tables 1 and 2 show that their accuracy is generally competitive. This actually suggests that, although PivNet learns every $k \in [1, k_{\max}]$ at the same time, it still effectively learns an accurate regression model. When PivNet-itr yields a less error (e.g.,
HEMPASS case) and applications need a single \( k \), PivNet-itr can be an option.

**Inference time.** We turn our attention to time to obtain \( k_{\text{max}} \)-NN distances. From Table 3, we first observe that Pivot is the fastest. This is reasonable, because it simply computes Equation (3), i.e., just "+" operation is required after computing \( \text{dist}(p,q) \). Recall that the accuracy of Pivot is the worst in most cases.

Second, QueryNet is faster than PivNet(-itr). This is also a natural observation. QueryNet does not use pivots, so the number of neurons in its input layer is smaller and it does not need to compute
We tested our $k$-NN distance estimation model in the application examples in Section 1. In this section, we set $k_{max} = 100$, and the hyper-parameters were the same as those in Section 4. We computed the exact results by using a $kd$-tree.

5 CASE STUDY

We next consider the distance-based outlier detection problems defined below.

**Definition 3** $(r, k)$-distance-based outlier detection problem \footnote{This is faster than other works \cite{5, 7, 37, 46}, because they incur $O(n^2)$ time.}.\label{eq:knn} Given a set $X$ of points, $k$, and a distance-threshold $r$, this problem is to detect all points $x \in X$ such that the number of points $x'$ satisfying $dist(x, x') \leq r$ is less than $k$.

**Definition 4** $(N, k)$-distance-based outlier detection problem \footnote{This is faster than other works \cite{5, 7, 37, 46}, because they incur $O(n^2)$ time.}.\label{eq:knn} Given a set $X$ of points, $k$, and $N$, this problem finds $N$ points $x \in X$ whose distances to their $k$-th nearest neighbor are the largest among the points in $X$.

Both problems can be solved by evaluating the distance to the $k$-th nearest neighbor for each $x \in X$.

**Setting.** In this study, we set $k = 50$. For the $(r, k)$-distance-based outlier detection ($(r, k)$-DOD) problem, we set $r$ so that the number of outliers is 1,000. Similarly, we set $N = 1,000$ for the $(N, k)$-DOD problem. To measure the accuracy, we used precision and recall. Let $TP$, $FP$, and $FN$ be the numbers of true positives, false positives, and false negatives, respectively. Precision and recall are defined as follows:

\[
\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}.
\]

For evaluation, we used PivNet-itr (PivNet) on HEPMASS, Household, PAMAP2, and Wisdom.

**Result.** Table 5 shows the accuracy of our estimation-based detection. We see that its detection accuracy is high (i.e., both $FP$ and $FN$ are small). In particular, we achieve almost perfect recall on HEPMASS, Household, and Wisdom. This result also confirms its high inference accuracy.

The detection time is shown in Table 6. Our approach is again much faster than $kd$-tree. The time complexity of $kd$-tree for retrieving $k$-NNs is $O(kn^{1-1/d})$, where $n = |X|$. It hence incurs $O(kn^{2-1/d})$ time for the distance-based outlier detection problems, and it takes a long time for DOD on large datasets, e.g., HEPMASS. On the
We next consider the estimation of a missing parameter of density-based clustering by utilizing PivNet. As density-based clustering, we focus on density-peaks clustering (DPC) \[42\]. This is a commonly used algorithm but already has many applications \[2\]. (For another famous density-based clustering DBSCAN, a reverse engineering-like tool has been already provided in \[13\].)

For each point \(x_i \in X\), DPC computes the local density \(\rho_i\), which is the number of points \(x_j\) such that \(\text{dist}(x_i, x_j) \leq d_{\text{cut}}\), where \(d_{\text{cut}}\) is a user-specified distance threshold. In addition, for each \(x_i \in X\), DPC computes its dependent point, which is the nearest point of \(x_i\) with higher local density than \(\rho_i\), and the dependent distance \(\delta_i\), which is the distance between \(x_i\) and its dependent point. As with Definition 3, DPC removes noises that have less local density than \(\rho_{\text{min}}\). It is intuitively seen that, if a non-noise point \(x\) has a large dependent distance, it is a density-peak in the space centered at \(x\). DPC employs this concept and regards it as a cluster center.

Consider the dataset illustrated in Figure 3(a). After we compute \(\rho_i\) and \(\delta_i\) for each \(x_i \in X\), \((\rho_i, \delta_i)\) can be mapped into a 2-dimensional space called a decision graph, see Figure 3. It visualizes that there are some points with large dependent distances, i.e., cluster centers. By setting a threshold \(\delta_{\text{min}}\) of dependent distance (e.g., \(\delta_{\text{min}} = 800\) in Figure 3(d)), DPC determines cluster centers, and the other non-noise points belong to the same clusters as their dependent points.

It is often the case that analysis (e.g., clustering) results are known but (some of) their parameters are missing \[25\]. In the context of DPC, it is possible that we have cluster labels and a decision graph but \(d_{\text{cut}}\) is missing. To reproduce the results, a straightforward approach is to run DPC while testing some values of \(d_{\text{cut}}\). However, DPC needs \(O(n^2)\) time \[2\] (on low-dimensional datasets), so running DPC multiple times is not promising. We alleviate this concern by estimating \(d_{\text{cut}}\) and consider how to achieve this.

For simplicity, let us assume that \(\rho_{\text{min}}\) and the number of noises (denoted by \(m\)) are known. (They are obtained from cluster labels and the decision graph.) From Section 5.2, we can consider that each noise point \(x\) has the top-\(m\) largest distance between \(x\) and its \(\rho_{\text{min}}\)-NN. Therefore, by using the top-\(m\) largest \(\rho_{\text{min}}\)-NN distance as \(d_{\text{cut}}\), we can (approximately) distinguish non-noise points from noises and then reproduce the clustering result.

To test this idea, we used the synthetic dataset Syn in Figure 3(a), which was generated based on a random walk model \[22\]. Syn has 200,000 points and eight density-peaks. The exact clustering result with \(d_{\text{cut}} = 200\) and its decision graph are respectively illustrated in Figures 3(b) and 3(d) (best viewed in color). We set \(\rho_{\text{min}} = 50\), and then Syn has 186 noises. Next, we used PivNet to estimate the \(\rho_{\text{min}}\)-NN distance for each point. As a result, the 186-th largest 50-NN distance was 179.23. We used this value as the estimated \(d_{\text{cut}}\) and ran DPC. The clustering result and decision graph are respectively illustrated in Figures 3(c) and 3(e), respectively. It can be seen that our idea reproduces the clustering result almost perfectly. This reproduction needed only one-time DPC and \(O(n \log n)\) time to estimate \(d_{\text{cut}}\) (\(O(n)\) time for \(\rho_{\text{min}}\)-NN distance estimation and \(O(n \log n)\) time to sort the distances), demonstrating that PivNet efficiently helps reverse engineering.
5.4 Approximate $k$-NN Search

Last, we apply our distance estimation to approximate $k$-NN search. In traditional $k$-NN search algorithms, the threshold is initialized at $\infty$, and it is updated during the search. If we can use a threshold that is (very) close to the exact $k$-NN distance at initialization, the search efficiency is accelerated (although it can produce an approximate result when it is smaller than the exact value). We tested this idea by using the datasets in Section 4. For each dataset, we randomly
sampled 10,000 points and used them as queries. We initialized the threshold by our estimation and ran $k$-NN search on a $kd$-tree. Table 7 shows the running time (inference time is also included) with $k = 25, 50, 75,$ and $100$. We achieved $1.2x$–$2.9x$ speed-up, compared with the exact retrieval. This result confirms that (i) the above idea is effective and (ii) our estimation does not yield a loose threshold.

Table 8 shows the average and median recall of the approximate $k$-NN search results. The average recall is generally high but is less than $1$. This means that, for some queries, the estimated thresholds were less than the exact values, yielding answer sets with less than $k$ points. However, this is easily alleviated. For example, if applications require $25$ nearest points, they can specify $k$ that is larger than $25$ (e.g., $k = 30$), since the time of the estimation-based approximate $k$-NN search does not change much even if $k$ increases a bit. It is also important to note that the estimation-based approximate $k$-NN search (i) is robust w.r.t. $k$ (which is consistent with the result in Section 4.2) and (ii) provides the exact result (with less running time) for many queries, see the median result.

6 RELATED WORK

Similarity search is a primitive operator for many applications, and efficient processing of range and $k$-NN queries has been significantly studied. Representative approaches are space partitioning ones, such as $kd$-tree and R-tree. Given a range or $k$-NN query, a branch-and-bound or best-first algorithm is conducted on such a tree to prune unnecessary nodes. However, when we want to know the distances to the $k$-NNs of the query, these approaches are not sufficient, since they involve many data accesses. For example, $kd$-trees incur $O(kn^{1-1/d})$ data accesses to retrieve $k$-NNs [47].

Recently, a concept “index as a model” [28] has been introduced to build indices of datasets, and learned indices have been devised. The first learned index RMI [28] was considered for one-dimensional data to alleviate the traversal costs of B+-trees. An RMI has some neural network models that estimate the position of a given query, and this approach can skip many data accesses. Some works introduced learned indices to multi-dimensional data [19, 29, 36, 38, 56]. However, their objectives are different from ours or their adaptability is limited. For example, literature [56] considered only 2-dimensional data. Literatures [19, 36] assume multi-attribute data and focus on relational range queries that search $x$ satisfying $(a \leq x[a] \leq b) \land \cdots \land (a' \leq x[d] \leq b')$, where $x[i]$ shows the $i$-th attribute of $x$. Although [29, 38] support (approximate) $k$-NN queries, their objective is to minimize disk I/O cost and the indices are specific to disk blocks. Their approaches are trivially not suitable for in-memory distance estimation, and Pivot in Section 4 is a better competitor as it needs only $O(1)$ time for our problem.

Cardinality/Selectivity estimation is also related to our work. In a nutshell, this problem estimates the output size of a range query. Query cost estimation and making a batch query processing schedule receive benefits from this problem. A classic approach that tries to accurately estimate the cardinality of a given query is multi-dimensional histogram [12, 23]. Given a query and a threshold, this approach finds the bin that corresponds to the query. Assuming that points in this bin follows a specific (e.g., uniform) distribution, this approach estimates the cardinality. However, real datasets generally do not follow a specific distribution, so the accuracy of this approach tends to be low.

To effectively adapt to real data distributions, ML-based cardinality estimation techniques were devised, e.g., in [24, 31, 45, 57]. Some works, e.g., [21], employ decision-tree-based models [18]. (Section 4 shows that PivNet beats the state-of-the-art decision-tree-based model.) Neural network models were also considered for cardinality estimation [45, 51, 52, 57], but they cannot provide multiple values as the output at a time. Moreover, it is not trivial to employ these cardinality estimation models in our problem.

7 CONCLUSION

Motivated by the importance of scaling proximity-based data analysis, we addressed the problem of $k$-NN distance estimation. As a solution for this problem, we proposed a machine-learning technique, PivNet. This is a neural network-based regression model that estimates the distances to the $k$-NNs of a given query at a time in $O(1)$ time. This model is trained not only from training queries but also from their nearest pivots to make it more accurate. We conducted experiments on real datasets. The results demonstrate that PivNet yields a small estimation error while keeping fast inference time. Moreover, we did case studies to investigate the usefulness of our model for existing proximity-based data analysis tools. Thanks to its accuracy and efficiency, we observed that it provides faster analysis with high accuracy.

This paper focused on low-dimensional data. It is natural to use dimensionality reduction for high-dimensional data cases, such as principal component analysis and t-SNE [33], in order to fit them into our assumption. However, dimensionality reduction may break the original global distribution, so there would exist applications that need to deal with high-dimensional data as it is. Estimation of $k$-NN distances for high-dimensional data is an open problem.

ACKNOWLEDGMENTS

This research is partially supported by JSPS Grant-in-Aid for Scientific Research (A) Grant Number 18H04095, JST PRESTO Grant Number JPMJPR1931, and JST CREST Grant Number JPMJCR21F2.

REFERENCES

[1] Daichi Amagata, Yusuke Arai, Sumio Fujita, and Takahiro Hara. 2022. Learned k-NN Distance Estimation. In SIGSPATIAL. 1:1–1:4.
[2] Daichi Amagata and Takahiro Hara. 2021. Fast Density-Peaks Clustering: Multicore-based Parallelization Approach. In SIGMOD. 49–61.
[3] Daichi Amagata, Makoto Onizuka, and Takahiro Hara. 2021. Fast and Exact Outlier Detection in Metric Spaces: A Proximity Graph-based Approach. In SIGMOD. 36–48.
[4] Daichi Amagata, Makoto Onizuka, and Takahiro Hara. 2022. Fast, Exact, and Parallel-friendly Outlier Detection Algorithms with Proximity Graph in Metric Spaces. The VLDB Journal (2022).
[5] Fabrizio Angiulli and Fabio Fassetti. 2009. DOLPHIN: An Efficient Algorithm for Mining Distance-based Outliers in Very Large Datasets. ACM Transactions on Knowledge Discovery from Data (TKDD) 3, 1 (2009), 1–57.
[6] Pierre Baldi, Kyle Cranmer, Taylor Faucett, Peter Sadowski, and Daniel Whiteson. 2016. Parameterized machine learning for high-energy physics. arXiv preprint arXiv:1601.07913 (2016).
[7] Stephen D Bay and Mark Schwabacher. 2003. Mining Distance-based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule. In SIGKDD. 29–38.
[8] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. Commun. ACM 18, 9 (1975), 509–517.
