A NEW HASHING BASED NEAREST NEIGHBORS SELECTION TECHNIQUE FOR BIG DATASETS

Jude Tchaye-Kondi, Yanlong Zhai and Liehuang Zhu

School of Computer Science, Beijing Institute of Technology, Beijing, China

ABSTRACT

KNN has the reputation of being a simple and powerful supervised learning algorithm used for either classification or regression. Although KNN prediction performance highly depends on the size of the training dataset, when this one is large, KNN suffers from slow decision making. This is because each decision-making process requires the KNN algorithm to look for nearest neighbors within the entire dataset. To overcome this slowness problem, we propose a new technique that enables the selection of nearest neighbors directly in the neighborhood of a given data point. The proposed approach consists of dividing the data space into sub-cells of a virtual grid built on top of the dataset. The mapping between data points and sub-cells is achieved using hashing. When it comes to selecting the nearest neighbors of a new observation, we first identify the central cell where the observation is contained. Once that central cell is known, we then start looking for the nearest neighbors from it and the cells around. From our experimental performance analysis of publicly available datasets, our algorithm outperforms the original KNN with a predictive quality as good and offers competitive performance with solutions such as KDtree.

KEYWORDS

Machine learning, Nearest neighbors, Hashing, Big data.

1. INTRODUCTION AND MOTIVATIONS

The K nearest neighbors or simply KNN is an algorithm that relies on a very simple principle: tell me who your neighbors are and I'll tell you who you are(Figure 1). Therefore, to make a prediction, KNN does not rely on a statistical model, it learns nothing from the training data and has to carry the full dataset during its decision making. For this reason, KNN is categorized as a Lazy Learning algorithm.

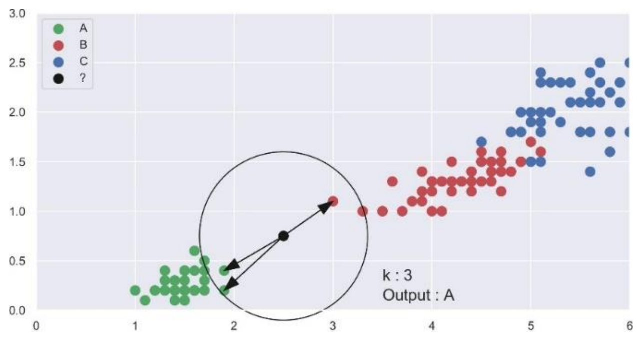

Figure 1: KNN example

The K of KNN is not a parameter but a hyperparameter because, unlike conventional parameters, it will not be learned automatically by the algorithm from the training data. It is up to us to optimize it, using the test dataset. To generate a prediction for a new observation x, the algorithm searches for the K closest instances to x from the dataset. For these instances, the algorithm will be based on their target values to generate the output of the observation we want to predict. Accordingly:

- If we are in a regression task, the prediction is the average or median of the K nearest instances' labels.
- If we are in a classification task, the prediction is commonly majority class among the K nearest instances' labels.

Figure 1 illustrates simple classification example with k = 3. We can clearly observe that among the 3 nearest data points, 2 belongs to class A. KNN will then output class A for the black data point. KNN is a supervised learning method with very good prediction accuracy, hence its wide use in several domains. In medicine, researchers proposed a KNN based drug classification approach used to categorize the different types of drug[1], it is also used for diagnosing Heart disease patients[2], cancer prediction and detection[3][4][5]. In computer vision, KNN work efficiently in image classification [6], face recognition [7]. The KNN algorithm is also present in cybersecurity where it is effectively used for credit card fraud detection[8], detection of intrusive attacks in a network system[9]. The strength of KNN is its simplicity and efficiency. However, behind this efficiency, it hides two big weaknesses that are:

- The large size of the final model: since KNN is not a statistical model, all training datasets must be carried out during inferences.
- Slow inferences: during inference, KNN must iterate through the dataset for distance calculations before selecting the nearest neighbors. This process has a time complexity of $(fn)log(fn)$ with n the training dataset size and f the number of features.

In addition, KNN is also very sensitive to noise (outliers), highly dependent on the choice of$K$ and the distance metric. The slowness and model size weaknesses restrict the use of KNN with large training data. Since KNN runs in memory and as the RAM is limited, it will be difficult to keep a large dataset in memory. Due to the slowness that results, this algorithm is not suitable for real-time applications or applications having strict time limits requirements. In this paper, we focused on the slowness problem of KNN during predictions with big datasets. To improve the prediction time efficiency, we are proposing a new hashing-based algorithm called GHN: Grid Hashing Neighborhood. GHN approach consists of splitting the data space into sub-cells by building a virtual grid on top of it. The virtual grid is constructed in such a way that each data point in the dataset can be mapped to a specific grid cell with a hash function. Both the virtual grid and the mapping hash function are constructed at the learning phase and then used during inferences to speed up the nearest neighbors selection. During prediction, the nearest neighbors of a new observation are selected in two steps. First, the central cell to which the new observation belongs is identified using the mapping hash function. Second, we search for nearest neighbors from this central cell and cells around it layer by layer. Therefore, unlike the native KNN, which have to go through the entire training dataset, GHN is able to select the nearest neighbors directly in the neighborhood of a new observation. Our performance analysis shows that our approach is faster in making predictions than the native KNN.

The rest of the paper is structured as follows. Section 2 reviews related work. Section 3 the methodology of GHN. Section 4 evaluates the performance of our implementation of GHN, the original KNN, and KDtree on some publicly available datasets. Finally, the paper is concluded in Section 5.

2. RELATED WORK

The slowness of KNN predictions is not a new problem in machine learning, unlike humans who, just by looking at the data representation in 2D or 3D vector space, can intuitively guess the nearest neighbors of a data point, a computer requires more calculations for the same task. Three main groups emerge among the different techniques used to compute the nearest neighbors that are: data reduction approaches, hashing based approaches, and tree-based approaches. Most of the literature's suggested solutions consist of data reduction strategies. These strategies try to reduce the size of the training data, therefore, reducing the number of distance calculations and the amount of memory needed by the model during prediction. Reducing the size of the training data can be effective for certain types of datasets that may still work accurately by only considering some special data points. Although they are quicker in prediction and enhance memory usage, these approaches are less adopted. It may be because data reduction usually does not lead to the same prediction accuracy as KNN.

In [10],[11],[12],[13] concave and convex hulls-based techniques are proposed and used to reduce each class samples to their edge data points. In these techniques, only the edge points are used in the training datasets for classification. Hart et Al. proposed the condensed nearest neighbor(CNN)[14] which reduces its data by selecting prototypes U from the training data in a way that 1NN with U can classify the samples almost as precisely as 1NN does with the dataset. CNN works in 3 steps [15]: 1) Scans all the elements of the training data X, looking for an element x whose nearest prototype from U has a label different from x. 2) Remove x from X and add it to U. 3) Repeat the operation until no other prototype is added to U. In the end use U instead of X to train the model. In the same perspective, Salvador et Al. introduce compressed kNN[16] which is a binary level data compression technique. The method proposes to compress observations into packets of a certain number of bits. In each packet, attributes are stored through binary level operations. This technique reduces the amount of RAM needed to maintain the training data in memory. An interesting feature of the compressed kNN approach is that the information can be decompressed, observation by observation on the fly and in real-time, without the need to decompress the entire dataset in memory. Unfortunately, compressed kNN also suffers from slowness and only works with categorical features.

During our research, we noticed that there were only a few researches attempts to use hashing techniques to estimate the nearest neighbors. Hashing can be used to group similar data points in buckets. The most popular hashing based solution is the LSH(Locality Sensitive Hashing) family [17] [18] [19] [20]. LSH based solutions use random plane projections in the data space to divide that space into sub-regions. These sub-regions are then used as a bucket to build a hash function. Even if LSH based solutions improve prediction time, the strategy behind them is ineffective and does not guarantee to get the real nearest neighbors hence its low adoption. Gao et Al. try to face this problem by suggesting another family of hashing technique, DHT[21], which, unlike the LSH family, can maintain relationships between the nearest neighbors. Tree-based solutions are the most adopted in real-world problems when it comes to approximating the nearest neighbors. The most famous are KD tree[22][23] and Ball Tree[24][25]. They are data structures that organize training data like a tree. When searching for the nearest neighbors, we navigate the tree from top to bottom, hoping that the region we led in will contain the nearest neighbors. Just like with LSH, these tree-based solutions can easily miss the real nearest neighbors and they are mostly recommended for low dimensional space since they don't perform well with multi-dimensions. In conclusion, there is still no efficient solution to accurately estimate the KNN that provides low computation and memory cost. The existing one suffers from drawbacks like accuracy degradation or the risk of having fake nearest neighbors. Our solution adopts a unique hashing-based approach that allows us to directly select our neighbors around the observation during prediction with relatively good performances.

3. Proposed Solution

Compared to the other machine learning algorithms KNN does not have a learning phase, the dataset does not undergo any transformation and is entirely maintained in memory. It is during predictions that KNN does all of its computations (distances, nearest neighbors selection). Unlike KNN, GHN has a learning phase before the prediction one. Figure 2 illustrates the GHN algorithm. We work with the simple case of a two-dimensional space, i.e. when the training data only have two features since it is much easier to visualize and understand. However, it can be generalized to multiple dimensions. Figure 2.a contains the observations of two classes, the class A observations are the blue squares, that of class B are the green circles, and the new observation whose class is to be predicted is represented by a red cross. For this example, $k = 3$, so our goal is to find the 3 nearest data points to our new observation. GHN algorithm consists mainly of two steps:

1) **Cells sampling:** this phase is accomplished during training. We subdivide the data space into identical sub-cells by building on top of it a virtual grid as illustrated in Figure 2.b. A Hash function is used to map training data points to their corresponding sub-cells.
2) **Exploration:** this phase consists of selecting the nearest neighbors of a given input. As shown in Figure 2.c, GHN firstly determines the sub-cell to which the new observation with unknown output belongs by using the mapping hash function. Secondly, it searchesnearest neighbors from data points in this central cell and cells in its neighborhood layers by layer.

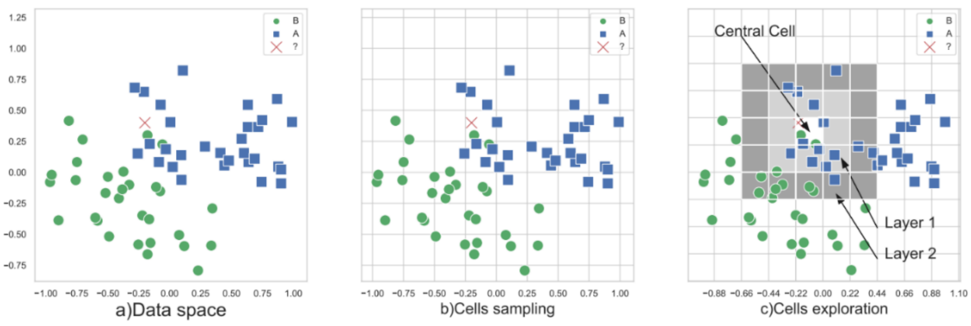

Figure 2: Different steps off GHN algorithm

3.1. Cell sampling

It is done during the model's training phase. As said above, during this phase, we build a virtual grid on top of the data space to split it into sub-cells. Each sub-cell will contain the data point located in the area it covers. These cells represent the buckets of our hash table, they are identical but not necessarily equilateral. Before building the virtual grid, we start by deciding the cell measurements on each dimension. To do this, we divide the values range covered by the training data points on each dimension in the largest possible number of splits. The division has to be done in a way that each of the splits we obtain ends up with at least one data point. The cell measurements on the corresponding dimension will then be the split width. The cell sampling process is illustrated with Figure 3, in which the first dimension can have a maximum of 7 splits and a maximum of 8 splits for the second dimension. The cell measurements will be respectively $(range1/7, range2/8)$ on the first and second dimensions. This way of determining the virtual grid's cell measurements ensures an optimal distribution of the data points in cells, it also facilitates the lookup of nearest neighbors during exploration. Once the measurements for each dimension are determined, Equation 1 defines the hash function that maps a data point to its corresponding cell. The result of Equation 1 uniquely identifies each cell of the virtual grid. Data points that belong to the same cell have the same cell id. GHN hash table does not keep any information about empty cells for memory efficiency. The entire cell sampling process is simplified by Algorithm 1.

Algorithm 1: Cell sampling Algorithm

```
grid : A hash table where the key is cellId;
a : cell measurements;
data : training data;
/* Model Training : Cells sampling                  */
for point in data do
    cellId = point//a;
    if cellId not grid then
        /* Initialize the cell                      */
        grid[cellId] = [];
    end
    grid[cellId].append(point);
end
```

$$cell\_id = P//a \qquad (1)$$

Where:

- P: The data point.
- $//$: Integer division.
- a: Cell Measurements

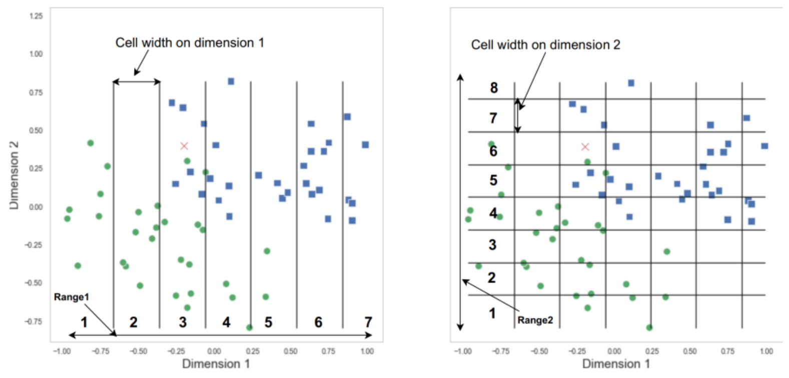

Figure 3: Cell measurements

3.2. Exploration

The actual selection of neighbors data points is made during this exploration step. The idea is to start with the cell to which our new observation belongs. Using Equation 1, we can get its id, compute the ids of cells in its neighborhood to search for the nearest data points. The two steps of the exploration are as follows:

1) Get the central cell, the one that contains the new observation using Equation 1.
2) Retrieve data points from the central cell and its neighbors layer by layer, as shown in the example of Figure 2.c.

We use the breadth-first search [26] (BFS) techniques in our implementation to compute ids and visit cells around the central one. When a cell is visited, the data points it contains are collected in a buffer. The buffer here is a heap of size k and only keeps the k potential nearest element using heap sort mechanism[27]. The exploration stops when a layer is visited and there is no update among the buffer elements. In Figure 2.c, $K = 3$, the three nearest data points are selected as follows:

- Central cell exploration: The central cell only contains 1 data point. We add this observation to our buffer, and we explore the cells on the first layer since the buffer size is less than 3.
- First layer Exploration: After visiting these 8 cells, from the data point collected from them, the buffer will only retain the 3 closest ones to our observation.
- Second layer Exploration: the second layer consists of 16 cells. Visiting these cells does not provide any update among the data points in the buffer, so the exploration stops there.

At the end of the exploration, the buffer will remain with the exact k elements that are our nearest neighbors. It is important to note that in some rare cases, the exploration may miss the real nearest neighbors due to the fact the virtual grid cells don't have the same measurements for each dimension. The entire exploration process is illustrated by Algorithm 2.

Algorithm 2: Exploration Algorithm

```
grid : The hash table ;
a : cell measurements;
data : training data;
/* Select K Nearest: Cells exploration */
central_cellId = new_input//a;
queue = [central_cellId];
buffer = [];
while queue is not empty do
    cellId = queue.dequeue()
    if cellId not grid then
        buffer.addAll(grid[cellId])
    end
    Add neighbors' cells to the queue;
    for next_cell in neighborsCells do
        queue.enqueue(next_cell)
    end
end
return buffer;
```

3.3. Performances Comparison with KNN

The complexity of GHN mainly depends on the number of features and the exploration depth (visited layers) that is strongly influenced by the data distribution. GHN can achieve record performance with large training datasets compared to existing solutions. The more the data grows, the faster is GHN since it directly searches for neighbors in the observations' neighborhood. Its performance is almost constant $O(1)$ when the input is located in a densely populated area of the data space, otherwise, GHN will require a little more effort and more exploration. The time taken by GHN to make a prediction is the time taken by its exploration phases, added to the time needed to process the buffer that contains the k nearest elements:

$$Time\ Complexity = Exploration + Buffer\ processing$$

- Exploration time: Depends on the exploration depth and the number of data points processed from visited cells. Knowing the exploration depth, Equation 3 can help to define the total number of visited cells.
- Buffer processing time: it takes $O(k)$ time to process the buffer because only k elements remain at the end of the exploration phase.

During the exploration, the number of cells on each layer given by the following Equation 2, is proved in Appendix A.

$$n = (2l + 1)^f - (2l - 1)^f \tag{2}$$

Where:

- l: the number of visited layers.
- f: the number of dimensions or features.

Therefore, if the exploration stops after l layers, the total number of cells from the central cell to the last layer is expressed by the following Equation 3. We also demonstrate it in Appendix B.

$$S_{cells}(l) = (2l + 1)^f - 1 \tag{3}$$

The number of features f is fixed and does not change while using the model, only the exploration depth l influences $S_{cells}(l)$. It is difficult to extrapolate the different values of the exploration depth and the number of data points processed during the exploration phase since they depend on the dataset. For this reason, it is difficult to compare GHN directly with the KNN. Nevertheless, we will rather look at the best-case comparison of them. The best case that GHN allows is when the exploration is done in a very dense area and stop after exploring a single layer ($depth = l = 1$) and collecting k data points. For this best-case scenario, the number of visited cells is calculated by setting $l = 1$ in Equation 3 gives $S_1 = 3^f - 1$.Then the best-case complexity of is:

$$O\left(3^f - 1\right) + O(k) \simeq O(1) + O(1) \simeq O(1)$$

- $O(3^f - 1)$ is the cell exploration complexity.f can be neglected since it is a constant: $O(3^f - 1) \simeq O(1)$.
- $O(k)$ is the complexity for processing buffer. k also is a constant and can be neglected.

The proposed approach can achieve the best-case complexity of almost $O(1)$, which is far better than the original KNN and which has not yet been possible with all the solutions proposed so far. With original KNN, the best and worst-case time complexity is $O(nflog(fn))$. Both GHN and KNN have a memory complexity of $O(fn)$. Although GHN uses slightly more memory than KNN to store its hash table, the number of cells will never exceed the size of the training dataset n. GHN is the only proposed solution whose prediction time performance is not negatively influenced by the growth in the training data size. With existing solutions, the more the dataset size increases the slower they are but with our approach, the more the learning data increases, the better.

3.4. Discussion

The proposed solution can be used for classifications as well as for regressions tasks as it is only intended to improve the selection of nearest neighbors. In the draft of Figure 4 we can notice that once the nearest neighbors selected, they are used as input of a classification model that is responsible for predicting the majority class by a vote or a regression model that will predict the average or the median of the k selected samples. Features in the training dataset may take their values from completely different scales of magnitude. It is recommended for the training data to be rescaled in order to set features on the same magnitude. This is carried out using data scaling techniques such as Standardization, Mean Normalization, Unit Vector, etc. GHN is not suitable for all types of datasets as it assumes that all data points converge in the same area of the data space. When the data points are far away from each other or when our observation is very far from the region where the data points are located, GHN's performance degrades. As KNN, GHN is also subject to the curse of dimensionality [28]. This occurs when the number of states exponentially increases for a tiny increase in the number of dimensions or parameters due to a combinatory explosion. The phenomena can be observed in Equation 3. Each time the number of layers increases, the number of cells to explore is an exponential function of the dimensions.

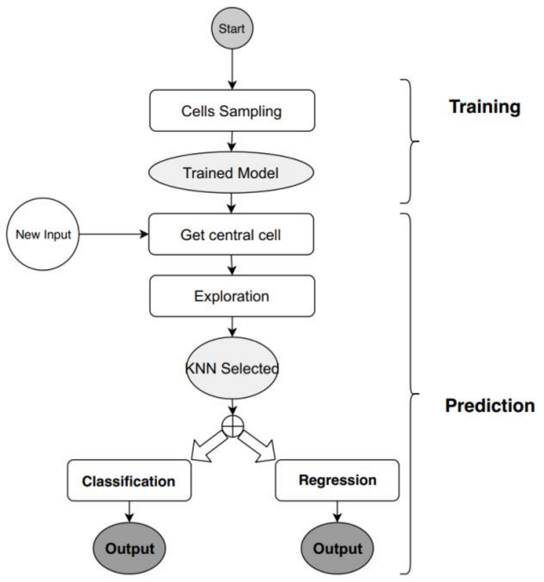

Figure 4: GHN Flow

4. EXPERIMENTS AND RESULTS

We evaluate the GHN performance against the original KNN and the popular KDTree. Our experiment target two main metrics, the prediction time which helps to evaluate the time efficiency, and the accuracy score which can tell us about how well the model is performing. Our tests are performed on 5 real dataset that we grab from different datasets repository [29], [30], [31], [32], [33], [34], these datasets are presented in Table 1. Each dataset is used for classification tasks only in order to facilitate comparisons. We scale the datasets and reduce their dimensions by using Principal component analysis(PCA). Our experiment is implemented in python 3.6 and the running environment is an Ubuntu laptop of processor Intel i7.

Table 1: Datasets

Datasets	Description
Fashion MNIST [34]	Fashion-MNIST is a dataset of Zalando's article images. Each example is a 28x28 grayscale image, associated with a label from 10 classes.
MNIST [30]	MNIST database of handwritten digits has a training set of 60,000 examples. The digits have been size-normalized and centred in a fixed-size image.
Pulsar Star [29], [33]	Describes a sample of pulsar candidates collected during the High Time Resolution Universe Survey.
Wine Quality [31]	Data about various chemical combination of red wine.
Russian Demography [32]	Russian Demography (1990-2017) Dataset. It contains demographic features like natural population growth, birth rate, population, etc.

We have collected in Table 2 the time taken by each model to evaluate the test data for each dataset.

Table 2: Prediction Times(ms) on various datasets

Datasets	GHN	KDTree	KNN
Fashion MNIST	84.29	52.40	228.77
MNIST	10.00	5.22	111.48
Pulsar Star	9.09	15.00	60.20
Wine Quality	0.27	0.87	2.46
Russian Demography	0.19	0.90	2.92

For each dataset, 80% is used for training and the remaining 20% for testing. In the results of Figure 5, GHN and KDtree are much faster than the original KNN on the 5 datasets. We also notice that KDT is slightly faster than GHN on image data type, this is due to the effect of the curse of dimensionality faced by GHN during the exploration since a flattened image end up with a high dimensional vector. This problem is mitigated by using PCA to bring the dimensions to a good balance between speed and accuracy. On the contrary, for other types of data, GHN is faster than KDtree. By analysing Table 2 data and Figure 5 results, GHN is the best choice for real-time applications if we must choose between GHN and KNN.

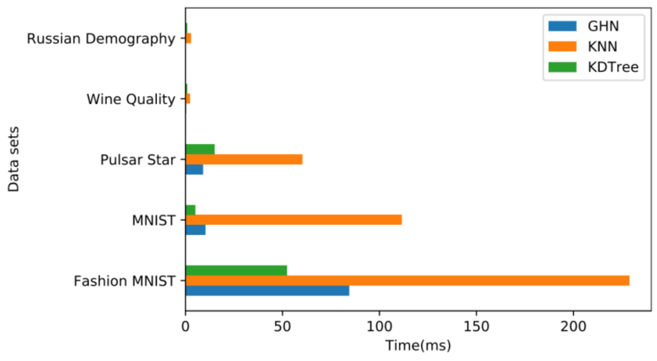

Figure 5: Comparison of prediction times on various datasets

In Table 3 and Figure 6, we compare accuracies of GHN, KDTree and KNN on all datasets.

Table 3: Accuracies on various datasets

Datasets	GHN	KDTree	KNN
Fashion MNIST	0.86	0.80	0.88
MNIST	0.74	0.74	0.74
Pulsar Star	0.99	0.99	0.99
Wine Quality	0.65	0.65	0.65
Russian Demography	0.50	0.49	0.52

With these results, we can conclude that all solutions accuracies are almost identical except for some slight variations. GHN offers better accuracy than KDtree on Russian Demography data and Fashion MNIST.

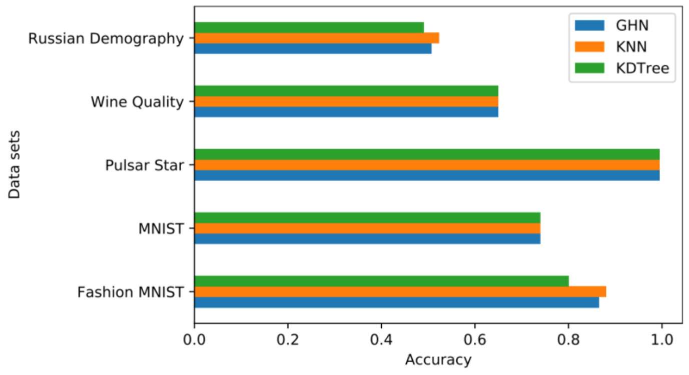

Figure 6: Comparison of accuracies on various datasets

Our experiment confirmed the fact that GHN can improve the time efficiency during predictions. It displays an accuracy as concurrent as that of KNN and KDTree. The main goal of GHN is to improve the prediction time by selecting the nearest neighbors directly in the neighborhood of our observation. Like KNN, GHN is sensitive to noise and is also subject to the curse of dimensionality. On the other hand, compared with proposed solutions till now that deal with the slowness of KNN, GHN is the only one capable of achieving almost constant time performance when the observations are in a densely populated area of the data space.

5. CONCLUSION

This paper proposes a new technique for picking the nearest neighbors that improve the prediction time for KNN, which turns out to be very slow. The algorithm works in two steps: Cell sampling and Exploration. During the first step, a hash function maps the data points with cells of a virtual grid built on top of the data space. Finally, the second step running during predictions consists of exploring and selecting the nearest data points. The experimental results validate the superior speed of GHN against KNN on all our testing datasets. GHN is compatible for both regression and classification tasks with a prediction efficiency as good as that of KNN. However, because the curse of dimensionality effect that exponentially increases the number of cells to explore from layer to layer, GHN is only recommended for low dimensional data spaces.

APPENDIX A

PROOF OF EQUATION 2

How many cells are on a given layer l? To answer this question,let's define by:

- a : cell measurements
- l : the layer
- f : the number of features

By looking at the Figure 2.c, we can deduce that the area from the central cell to a layer l is:

$$A(l) = [(2l + 1)a]^f$$

To only obtain the area covered by cells on layer l only, we must, therefore, deduct from $A(l)$ the area $A(l - 1)$:

$$A(l) - A(l-1) = [(2l + 1)a]^f - [(2(l - 1) + 1)a]^f$$

Now we can compute the number of cells on layer l. For that we just have to divide $A(l) - A(l - 1)$ by the cell volume a^f:

$$N_{cells} = \frac{A(l) - A(l - 1)}{a^f}$$

$$N_{cells} = \frac{[(2l + 1)a]^f - [(2(l - 1) + 1)a]^f}{a^f}$$

$$N_{cells} = (2l + 1)^f - (2l - 1)^f$$

Hence Equation 2.

APPENDIX B

PROOF OF EQUATION 3

In this Appendix, we want to find the number of cells from the first layer to a layer l.

We obtain the area from the first to layer l by deducing from $A(l)$ defined in Appendix A the volume of the central cell:

$$A(l) - a^f = [(2l + 1)a]^f - a^f$$

Dividing this volume with the cell area gives us the total number of cells:

$$S_{cells}(l) = \frac{A(l) - a^f}{a^f}$$

$$S_{cells}(l) = \frac{[(2l + 1)a]^f - a^f}{a^f}$$

$$S_{cells}(l) = (2l + 1)^f - 1$$

Which gives Equation 3.

ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers for their insightful suggestions. This research work is supported by the Natural Key R&D of China (No.2018YFC0830104).

REFERENCES

[1] J. Akhil, B. Deekshatulu, and P. Chandra, "Classification of heart disease using k- nearest neighbor and genetic algorithm," Procedia Technology, vol. 10, p. 85–94, 12 2013.

[2] M. Shouman, T. Turner, and R. Stocker, "Applying k-nearest neighbour in diagnosing heart disease patients," International Journal of Information and Education Technology, vol. 2, pp. 220–223, 01 2012.

[3] S. A. Medjahed, T. A. Saadi, and A. Benyettou, "Breast cancer diagnosis by using k-nearest neighbor with different distances and classification rules," International Journal of Computer Applications, vol. 62, no. 1, 2013.

[4] K. Machhale, H. B. Nandpuru, V. Kapur, and L. Kosta, "Mri brain cancer classification using hybrid classifier (svm-knn)," in 2015 International Conference on Industrial Instrumentation and Control (ICIC), May 2015, pp. 60–65.

[5] M. Jabbar, "Prediction of heart disease using k-nearest neighbor and particle swarm optimization," Biomed. Res, vol. 28, no. 9, pp. 4154–4158, 2017.

[6] G. Amato and F. Falchi, "knn based image classification relying on local feature similarity," in Proceedings of the Third International Conference on SImilarity Search and APplications. ACM, 2010, pp. 101–108.

[7] H. Ebrahimpour and A. Kouzani, "Face recognition using bagging knn," in International Conference on Signal Processing and Communication Systems (ICSPCS'2007) Australia, Gold Coast, 2007, pp. 17–19.

[8] V. R. Ganji and S. N. P. Mannem, "Credit card fraud detection using anti-k nearest neighbor algorithm," International Journal on Computer Science and Engineering, vol. 4, no. 6, pp. 1035–1039, 2012.

[9] Y. Liao and V. R. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection," Computers & security, vol. 21, no. 5, pp. 439– 448, 2002.

[10] W. M. Getz and C. C. Wilmers, "A local nearest-neighbor convex-hull construction of home ranges and utilization distributions," Ecography, vol. 27, no. 4, pp. 489–505, 2004. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10. 1111/j.0906-7590.2004.03835.x

[11] Z. Szymanski and M. Dwulit, "Improved k-nearest neighbor´ classifier for biomedical data based on convex hull of inversed set of points," in Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2010, R. S. Romaniuk, Ed., vol. 7745, International Society for Optics and Photonics. SPIE, 2010, pp. 324 – 331. [Online]. Available: https://doi.org/10.1117/12.873054

[12] A. Moreira and M. Santos, "Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points." 01 2007, pp. 61–68.

[13] J.-S. Park and S.-J. Oh, "A new concave hull algorithm and concaveness measure for n-dimensional datasets," Journal of Information Science and Engineering, vol. 29, pp. 379–392, 03 2013.

[14] P. Hart, "The condensed nearest neighbor rule (corresp.)," IEEE transactions on information theory, vol. 14, no. 3, pp. 515–516, 1968.

[15] k-nearest neighbors algorithm. https://en.wikipedia.org/wiki/Knearest neighbors algorithm. Accessed: 2020-1-10.

[16] J. Salvador, Z. Ruiz, and J. Garcia, "Compressed knn: K-nearest neighbors with data compression," 2019.

[17] J. Pan and D. Manocha, "Fast gpu-based locality sensitive hashing for k-nearest neighbor computation," in Proceedings of the 19th ACM SIGSPATIAL international conference on advances in geographic information systems. ACM, 2011, pp. 211–220.

[18] Y.-M. Zhang, K. Huang, G. Geng, and C.-L. Liu, "Fast knn graph construction with locality sensitive hashing," in Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2013, pp. 660–674.

[19] S. Bagui, A. K. Mondal, and S. Bagui, "Improving the performance of knn in the mapreduce framework using locality sensitive hashing," International Journal of Distributed Systems and Technologies (IJDST), vol. 10, no. 4, pp. 1–16, 2019.

[20] G. Wu, Z. Zhao, G. Fu, H. Wang, Y. Wang, Z. Wang, J. Hou, and L. Huang, "A fast knn-based approach for time sensitive anomaly detection over data streams," in International Conference on Computational Science. Springer, 2019, pp. 59–74.

[21] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi, "Dsh: data sensitive hashing for high-dimensional k-nnsearch," in Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014, pp. 1127–1138.

[22] R. F. Sproull, "Refinements to nearest-neighbor searching inkdimensional trees," Algorithmica, vol. 6, no. 1-6, pp. 579–589, 1991.

[23] K. Zhou, Q. Hou, R. Wang, and B. Guo, "Real-time kdtree construction on graphics hardware," ACM Trans. Graph., vol. 27, no. 5, pp. 126:1–126:11, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1409060.1409079

[24] S. M. Omohundro, Five balltree construction algorithms. International Computer Science Institute Berkeley, 1989.
[25] T. Liu, A. W. Moore, and A. Gray, "New algorithms for efficient high-dimensional nonparametric classification," Journal of Machine Learning Research, vol. 7, no. Jun, pp. 1135–1158, 2006.
[26] A. Elmasry, T. Hagerup, and F. Kammer, "Space-efficient basic graph algorithms," 2015.
[27] R. Schaffer and R. Sedgewick, "The analysis of heapsort," Journal of Algorithms, vol. 15, no. 1, pp. 76–100, 1993.
[28] A. Hinrichs, E. Novak, and H. Wozniakowski, "The curse of˝ dimensionality for monotone and convex functions of many variables," arXiv preprint arXiv:1011.3680, 2010.
[29] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[30] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist, vol. 2, 2010.
[31] Data tells various chemical combination of redwine. https:// www.kaggle.com/sh6147782/winequalityred. Accessed: 2020-1-29.
[32] Russian demography data (1990-2017). https://www.kaggle.com/dwdkills/russian-demography. Accessed: 2020-1-29.
[33] Predicting a pulsar star. https://www.kaggle.com/pavanraj159/ predicting-a-pulsar-star. Accessed: 2020-1-29.
[34] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

AUTHORS

Jude Tchaye-Kondi: Received the BS degree from the Department of Computer Science, Catholic University of West Africa. He joins in 2017 Beijing Institute of Technology, China as a graduate student and is now pursuing a Ph.D. program. His research interest includes parallel and distributed computing, edge computing, machine learning. Currently, his research projects focus on applying artificial intelligence algorithms in the edge computing environment.

Yanlong Zhai: Received the B.Eng. degree and Ph.D. degree in computer science from Beijing Institute of Technology, Beijing, China, in 2004 and 2010. He is an Assistant Professor in the School of Computer Science, Beijing Institute of Technology. He was a Visiting Scholar in the Department of Electrical Engineering and Computer Science, University of California, Irvine. His research interests include cloud computing and big data.

Liehuang Zhu: Received the B.Eng. and Master Degrees in computer application from Wuhan University, Wuhan, Hubei, China, in 1998 and 2001 respectively. He received the Ph.D. degree in computer application from Beijing Institute of Technology, Beijing, China, in 2004. He is currently a Professor in the Department of Computer Science, Beijing Institute of Technology, Beijing, China. He is selected into the Program for New Century Excellent Talents in University from Ministry of Education, China. His research interests include internet of things, cloud computing security, internet, and mobile security.