# TUKNN: A Parallel KNN Algorithm to Handle Large Data

Parthajit Borah(✉), Aguru Teja, Saurabh Anand Jha, and Dhruba K. Bhattacharyya

Department of Computer Science, Tezpur University, Tezpur, Assam, India
parthajit@tezu.ernet.in

**Abstract.** In this work, we study the performance of the K-Nearest Neighbour (KNN) based predictive model in sequential as well as parallel mode to observe its performance both in terms of accuracy and execution time. We propose a parallel KNN algorithm, called TUKNN to handle voluminous data. Based on our experimental study, it has been observed that our method is capable of handling datasets with large dimensionality and instances with high accuracy. We also recommend best possible proximity measure and optimal range of $K$ values for better accuracy.

**Keywords:** Supervised · Feature · Parallel · Classification · Optimal

## 1 Introduction

With the proliferation of data being generated, there is an urgent need of new technologies and architectures to make possible to extract valuable information from it by capturing and analysis process. New sources of data include various sensor enabled devices like medical devices, IP cameras, video surveillance cameras, and set-top boxes, which contribute largely to the volume of big data. Due to data proliferation, it is predicted that 44 zettabytes or 44 trillion gigabytes of data will be generated annually by the end of 2020[1]. The data are continuously generated by the sources from internet applications and communications which are of large size, different variety, structured or unstructured, which is referred to as Big data. Big Data is characterized by three particularly significant V's -Volume, Velocity, and Variety. The term Volume signifies the plethora of data produced from time to time by various different organizations and institutes. Velocity characterizes the rate at which data is generated from different sources. The third V, Variety denotes the diverse forms of data which may be structured, semi-structured or unstructured, generated from several organizations. For example, data can be in the form of video, image, text, audio, etc. Apart from the mentioned characteristics above, two other key features are–incremental and dispersed nature. They are incremental in the sense that there is dynamic addition of new incoming data to the pile of big data. Big data are dispersed in nature because they are geographically distributed across different data centers. These are some of the distinguishing characteristics which sets big data apart from traditional

[1] https://www.emc.com/leadership/digital-universe/2014iview/index.htm.

R. Patgiri et al. (Eds.): BigDML 2019, CCIS 1317, pp. 1–13, 2020.
https://doi.org/10.1007/978-3-030-62625-9_1

databases or data-warehouses. The traditional data storage techniques are not adequate to store and analyze those huge volume of data. In short, such a data is so large and complex that most traditional data management tools are inadequate to store or process it efficiently.

There are various challenges associated with big data. Such a large volume of data if processed sequentially it takes lot of time. Second, how do we process and extract valuable information from the huge volume of data within the given timeframe? To address the challenges, it is required to know various computational complexities, information security, and computational method, to analyze big data. For example, many statistical methods that perform well for small data size do not scale to voluminous data. Similarly, many computational techniques that perform well for small data face significant challenges in analyzing big data. Big data analytics is the use of advanced analytic techniques against very large, diverse data sets that include structured, semi-structured and unstructured data, from different sources, and in different sizes from terabytes to zettabytes.

Predictive analysis gives a list of solutions by establishing the previous data patterns for a given situation. It studies the present as well as the past data and predict what may happen in the future or gives the probability what would happen in the future. We need to make use of such large data in order to make decisions in future. However, traditional machine learning and statistical methods in sequential mode takes much longer time in order to make prediction, especially, in case of intrusion data [3]. In this work, a traditional machine learning model–KNN with various proximity measures is studied both in sequential and parallel manner.

The major contribution of this paper is a parallel version of the KNN algorithm referred here as TUKNN. We also conduct an exhaustive experimental study on a good number of proximity measures in the KNN framework and recommend the best possible measure to achieve best effective, better accuracy with TUKNN algorithm. Further, we also recommend an optimal range for ‘k’ values to achieve best possible performance.

## 2 Related Work

KNN is a non-parametric classification method, which is simple but effective in many cases [5]. It classifies objects based on the closest training example in the feature space. For any test object *t* which is to be classified, its K nearest neighbors are retrieved, and this forms the neighborhood of the object *t*. Then based on a majority voting among the neighbours, the class label of *t* is decided.

In [9], the authors use the CUDA (Compute Unified Device Architecture) thread model to implement a CUDA based KNN algorithm. Adult data from UCI Machine Learning Repository were used to compare the performance of CUDA based implementation on GPU with ordinary CPU based implementation and authors suggests that KNN method is efficient for applications with large volume of data.

In [8], the authors implement CUKNN algorithm which constructs two multi- thread kernels such as distance calculation kernel and sorting kernel. With CUKNN, the authors claims that the method could achieve 15.2 times better execution time performance than CPU.

In [2], the authors propose a fast and parallel KNN algorithm and show the impact on content-based image retrieval applications. The authors implement the parallel version of KNN in C and MATLAB using GPU with CUDA.

## 3 Proposed Work

KNN is a widely used classification algorithm and can be considered parallel friendly because of the number of independent operations. When the training and testing datasets are large, then the speed of execution becomes quite slow which makes it suitable for parallel implementation. In this work, we implement KNN on CUDA framework. The proposed framework is depicted in Fig. 1. In our framework, we explore a good no of proximity measures in parallel during the mining process to recommend the best possible measure for better accuracy. The measures used are: Euclidean distance, Manhattan distance, Kulczynski distance, cosine similarity, Chebyshev Distance, Soergel distance, Sorensen, and Tanimoto.

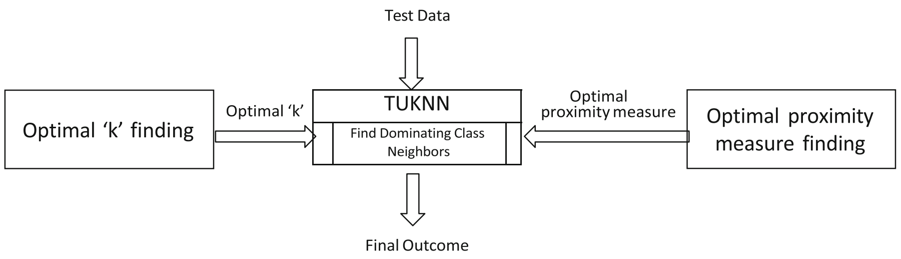

Test Data
Optimal 'k' finding
Optimal 'k'
TUKNN
Find Dominating Class Neighbors
Optimal proximity measure
Optimal proximity measure finding
Final Outcome

**Fig. 1.** Framework of the proposed work

### 3.1 Distance Measures

Dissimilarity is an essential component in the KNN algorithm. It influences the performance of the algorithm significantly in terms of speed and accuracy. Since, every proximity (similarity or dissimilarity) measure has its own advantages and disadvantages. So, we conduct an empirical study to evaluate their performance and subsequently to recommend the best possible measure for cost effective performance with TUKNN. Table 1 shows the distance measures and their mathematical expressions used in our work.

Further, we also carry out an exhaustive experimentation on large no of datasets by varying the $K$ values to identify an optimal range of $K$ values for best possible performance by TUKNN. Next, we present both sequential and parallel version of KNN algorithm.

### 3.2 Sequential KNN Algorithm

[1] For every fold in the 5 folds perform the steps 2 to 8.

**Table 1.** Distance measures and their mathematical expressions [4, 11]

| Measure & References | Math Expression |
|---|---|
| Euclidean | $D_{xy} = \sqrt{(\sum_{i=1}^{m}(x_i - y_i)^2)}$ |
| Manhattan | $D_{Manhattan}(x,y) = \|x_i - y_i\|$ |
| Kulczynski | $D_{kulczynski}(x,y) = \frac{(\sum \|x_i - y_i\|)}{(\sum max(x_i, y_i))}$ |
| Chebyshev | $D_{Chebyshev}(x,y) = max_i(\|x_i - y_i\|)$ |
| Cosine | $Sim(A,B) = cos(\theta) = \frac{A.B}{\|\|A\|\|\|\|B\|\|}$ |
| Sorgel | $D_{sg} = \frac{\sum_{i=1}^{d}\|P_i - Q_i\|}{\sum_{i=1}^{d} min(P_i, Q_i)}$ |
| Sorenson | $D_{soresnosn} = \frac{2\|x.y\|}{\|x\|^2 + \|y\|^2}$ |
| Tanimoto | $D_{tanimoto} = \frac{x.y}{\|x\|*\|x\|+\|y\|*\|y\|-x.y}$ |

[2] Split the dataset into test set and training set using 5-fold cross validation.
[3] For every test instance in the test set perform the steps 4 to 8.
[4] Find the distance between this test instance and all the training instances in the training set.
[5] Now, from the distances obtained from the step 4, find the first maximum *K* number of minimum values and thereby save the respective training instances having those values. Here, the maximum *K* value in the range (of K values) is chosen for the algorithm.
[6] For every *K* in a range of values perform the steps 7 & 8.
[7] Find the first *K* neighbors (i.e. the first K training instances with the minimum distances) from the results obtained in the step 5.
[8] Perform a majority voting among these neighbors; the dominating class label in the pool will become the class label of the test instance.

In step 5, instead of applying a sorting algorithm, we find the first *K* minimum distances and their respective training instances. This has been done in order to decrease the time complexity of the algorithm as the best sorting algorithm (Quick sort) takes O($N^2$) time where finding the first *K* minimum distances takes O($NK_{max}$) time. Here, N represents the size of the input (training set) and $K_{max}$ is the maximum *K*-Value in a range chosen for the algorithm.

### 3.3 The Proposed Parallel KNN Algorithm

The algorithm for parallel KNN implementation is stated below.

[1] For every fold in the 5 folds perform the steps 2 to 8.
[2] Split the dataset into test set and training set using 5-fold cross validation.
[3] For every *n* instances (2500) in the test set, perform the steps 4 to 8.
[4] Compute the distances between these *n* instances and all the training instances in the training set simultaneously by invoking the GPU kernel.

[5] Now, from the distances obtained from the step 4, find the first maximum *K* number of minimum values and thereby save the respective training instances having those values. The maximum K is the maximum K value in the range of K values chosen for the algorithm. This step is performed for all these *n* instances simultaneously with the help of the GPU kernel.
[6] For every K, perform the steps 7 & 8.
[7] Find the first *K* neighbors (i.e. the first *K* training instances with the minimum distances) from the results obtained in the step 5.
[8] Perform a majority voting among these neighbors and the dominating class label in the pool will become the class label of the test instance. The steps 6, 7 and 8 are performed for all these *n* instances simultaneously by invoking the GPU kernel.

## 4 Implementation and Results

For the parallel KNN, we compute all the distances between a set of test instances and all the training instances simultaneously. Hence, all the distances are computed in parallel at once. To calculate distance between the test instances and all the training instances in parallel, we use many cores of the GPU platform and develop the kernels in CUDA to compute the task in parallel. The most crucial task for a KNN classifier is to compute the distance *d* for finding the nearest neighbors. We implement the distance computation i.e. *d* on GPU platform which has resulted considerable improvement in the KNN performance.

The graphics card used in our work is NVIDIA Tesla k40c GPU Accelerator which has a memory of 12 GB. So, with a memory of 12 GB, we are able to compute the distance between 2500 test instances and all the training instances in the training set simultaneously on the GPU.

### 4.1 Datasets Used

We perform our experimentation on the following three types of datasets.

1. *Ransomware Dataset:* For our experiment, we use a dataset from Sgandurra et al. [10]. The dataset has total 582 and 942 instances of ransomware and goodware respectively. The 582 instances of ransomware comprise of 11 different variants. Also, it has total 30,692 features which collectively represent the characteristics of both goodware and ransomware. A detailed description of the dataset is given in the Table 2.
2. *SWaT Dataset:* Secure Water Treatment [1] (SWaT) data set is also used in our experimentation. The dataset contains a total of 946,722 instances out of which 54,620 instances belong to *attack* category. The dataset has 51 attributes and two labels namely *attack* and *normal*.
3. *UCI datasets*: A total of 20 datasets is also used in our work. The list of datasets used are given in the Table 3.

**Table 2.** Ransomware dataset characteristics

| Sl no | Class | No of samples |
|---|---|---|
| 1 | Goodware | 942 |
| 2 | Critroni | 50 |
| 3 | CryptLocker | 107 |
| 4 | CryptoWall | 46 |
| 5 | KOLLAH | 25 |
| 6 | Kovter | 64 |
| 7 | Locker | 97 |
| 8 | MATSNU | 59 |
| 9 | PGPCODER | 4 |
| 10 | Reveton | 90 |
| 11 | TeslaCrypt | 6 |
| 12 | Trojan-Ransom | 34 |
| | | Total samples: 1524 |
| | | Total features: 30962 |

## 4.2 Results and Observation

In our framework, an optimal range for $K$ values is determined based on experimental study on twenty datasets from UCI Machine Learning repository. This testing reduces the overhead of calculating the best possible K values for highest accuracy and makes our model faster. As we can see form the Table 4, in the majority cases (15 out of 20) the results show optimal $K$ values within the range of 2–9. Table 5 shows the ratio of CPU and GPU execution time for all the datasets used in our work. The optimal $K$ value of each proximity measures for which highest accuracy is obtained is reported in Table 6, 7 and 8.

### 4.2.1 Accuracy Comparison

The graph plots for the accuracy comparison for three datasets are shown below.

1. *Accuracy of Binary Class Ransomware Dataset:* The classification accuracy of KNN algorithm with all the eight distance measures of ransomware dataset with binary class is shown in Fig. 2. As shown in the figure, the highest accuracy i.e., 95.27 is obtained with *Kulczynski*, *Soergel*, *Sorenson*, and *Tanimoto* measures.
2. *Accuracy of Multi Class Ransomware Dataset:* In this study, our observation from Fig. 3 is that 82.32 is the highest accuracy given by KNN with *Kulczynski* measure.

**Table 3.** Characteristics of 20 datasets obtained from UCI repository

| S.I. | Dataset name | No of instances | No of features |
|---|---|---|---|
| 1 | Absenteeism at work | 740 | 21 |
| 2 | Audit | 777 | 18 |
| 3 | Banknote authentication | 1372 | 5 |
| 4 | Blood transfusion | 748 | 5 |
| 5 | Cardiotocography | 2126 | 23 |
| 6 | Diabetic debrecen | 1151 | 20 |
| 7 | Ecoli | 336 | 8 |
| 8 | Glass identification | 214 | 10 |
| 9 | Haberman | 306 | 3 |
| 10 | Hill valley | 606 | 101 |
| 11 | ILPD | 583 | 10 |
| 12 | Image segmentation | 2310 | 19 |
| 13 | Immunotherapy | 90 | 8 |
| 14 | Ionosphere | 351 | 34 |
| 15 | Iris | 150 | 4 |
| 16 | Libras | 360 | 91 |
| 17 | LSVT | 126 | 309 |
| 18 | Parkinson | 756 | 754 |
| 19 | Sonar | 208 | 60 |
| 20 | Soya bean | 47 | 35 |

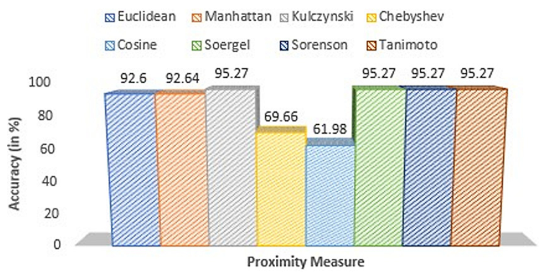


**Fig. 2.** Accuracy of ransomware dataset (binary class)

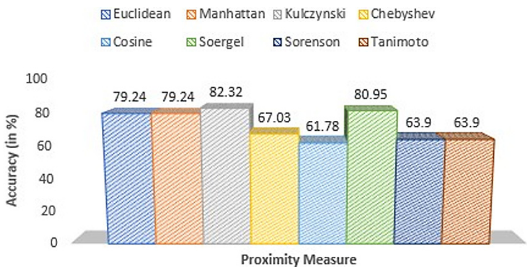


**Fig. 3.** Accuracy of ransomware dataset (multi class)

3. *Accuracy of SWaT Dataset:* Based on our study as depicted in Fig. 4 is that the model give same accuracy for all the eight measures i.e., 94.1 for this dataset. However, a difference in performance has been observed after 4*th* decimal (not reported here).

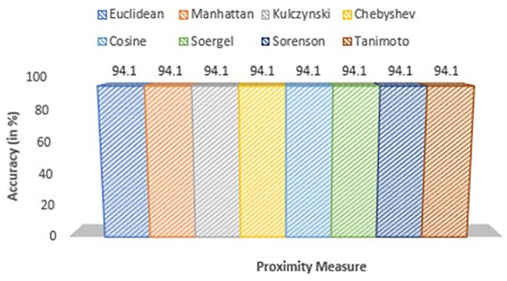
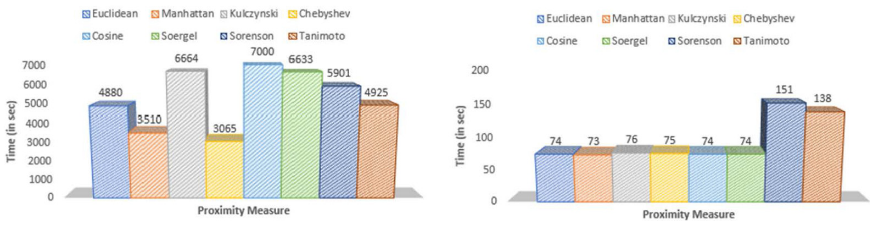


**Fig. 4.** Accuracy of SWaT dataset

### 4.2.2 Comparison of KNN and TUKNN in Terms of Execution Time

(a) *KNN vs TUKNN Time Comparison for Binary Ransomware Dataset:* The execution time comparison of KNN and TUKNN for binary ransomware dataset is shown in Fig. 5. It is clear from the figure that TUKNN performance is significantly better than KNN.

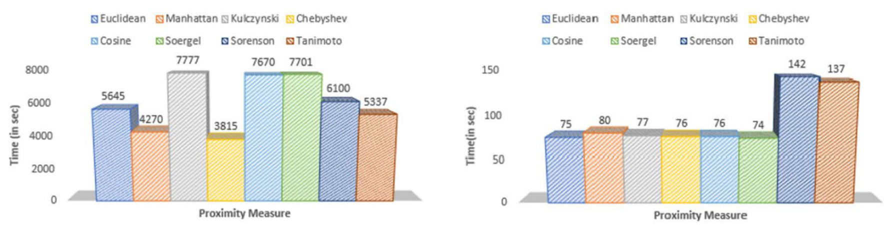


(a) a) (b) b)

**Fig. 5.** Time comparison for 2-class ransomware dataset: a) KNN and b) TUKNN

**Table 4.** *K* values to achieve maximum accuracy

| S.I. | Dataset name | No of instances | No of features | Value of K | Max Avg accuracy |
|---|---|---|---|---|---|
| 1 | Absenteeism at work | 740 | 21 | 8 | 30.20% |
| 2 | Audit | 777 | 18 | 3 | 93.70% |
| 3 | Banknote authentication | 1372 | 5 | 4 | 100% |
| 4 | Blood transfusion | 748 | 5 | 8 | 76.50% |
| 5 | Cardiotocography | 2126 | 23 | 37 | 98.40% |
| 6 | Diabetic debrecen | 1151 | 20 | 8 | 67.40% |
| 7 | Ecoli | 336 | 8 | 8 | 79.00% |
| 8 | Glass identification | 214 | 10 | 17 | 53.40% |
| 9 | Haberman | 306 | 3 | 39 | 77.40% |
| 10 | Hill valley | 606 | 101 | 3 | 54.70% |
| 11 | ILPD | 583 | 10 | 49 | 70.90% |
| 12 | Image segmentation | 2310 | 19 | 2 | 65.20% |
| 13 | Immunotherapy | 90 | 8 | 5 | 78.60% |
| 14 | Ionosphere | 351 | 34 | 3 | 83.20% |
| 15 | Iris | 150 | 4 | 2 | 96.00% |
| 16 | Libras | 360 | 91 | 3 | 11.50% |
| 17 | LSVT | 126 | 309 | 50 | 65.90% |
| 18 | Parkinson | 756 | 754 | 10 | 74.60% |
| 19 | Sonar | 208 | 60 | 4 | 46.30% |
| 20 | Soya bean | 47 | 35 | 2 | 98.00% |

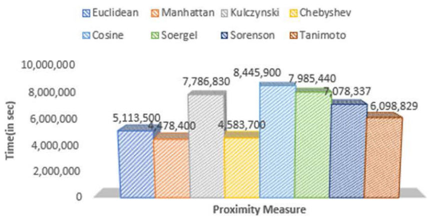


(a) c) (b) d)

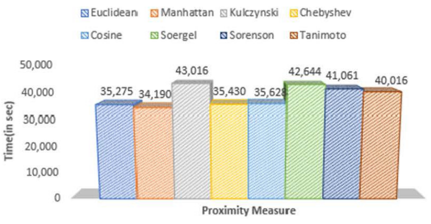
**Fig. 6.** Time comparison for multi-class ransomware dataset: c) KNN and d) TUKNN

(b) *KNN vs TUKNN Time Comparison for Multi-Class Ransomware Dataset:* Fig. 6 shows the performance comparison of KNN and TUKNN for multi class ransomware dataset. It is quite clear that TUKNN performance is much better than the KNN.

(c) *KNN vs TUKNN Time Comparison for SWaT Dataset:* In Fig. 7, it is clear that TUKNN implementation is significantly advantageous over KNN for SWaT dataset.



(a) e)



(b) f)

**Fig. 7.** Time comparison for SWaT dataset: e) KNN and f) TUKNN

**Table 5.** Ratio of CPU and GPU time (in seconds)

| S. no | Proximity measures | Ransomware dataset | | SWaT dataset |
|---|---|---|---|---|
| | | Binary class | Multi class | Binary class |
| 1 | Euclidean distance | 65.94 | 75.2 | 144.96 |
| 2 | Manhattan distance | 48.94 | 59.62 | 130.98 |
| 3 | Chebyshev distance | 40.86 | 50.19 | 129.37 |
| 4 | Cosine similarity | 94.59 | 100.9 | 237.05 |
| 5 | Kulczynski distance | 87.49 | 100.94 | 181.95 |
| 6 | Soergel distance | 89.63 | 104.06 | 187.25 |
| 7 | Sorenson distance | 39.07 | 42.95 | 172.38 |
| 8 | Tanimoto distance | 35.68 | 38.95 | 152.41 |
| 9 | Motyka distance | 70.91 | 77.21 | 193.68 |
| 10 | Ruzicka distance | 56.71 | 51.85 | 163 |

**Table 6.** Optimal ‘K’ values for proximity measures for 2-class ransomware dataset

| Proximity measure | Optimal ‘K’ value | Accuracy |
|---|---|---|
| Euclidean | 3 | 92.64 |
| Manhattan | 3 | 92.64 |
| Kulczynski | 3 | 95.27 |
| Chebyshev | 2 | 69.66 |
| Cosine | 7 | 61.9 |
| Soergel | 3 | 95.27 |
| Sorenson | 3 | 95.27 |
| Tanimoto | 3 | 95.27 |

**Table 7.** Optimal ‘*K*’ values for proximity measures for *n*-class ransomware dataset

| Proximity measure | Optimal ‘K’ value | Accuracy |
|---|---|---|
| Euclidean | 2 | 79.24 |
| Manhattan | 2 | 79.24 |
| Kulczynski | 3 | 82.32 |
| Chebyshev | 2 | 67.03 |
| Cosine | 8 | 61.78 |
| Soergel | 2 | 80.95 |
| Sorenson | 2 | 63.88 |
| Tanimoto | 2 | 63.88 |

**Table 8.** Optimal ‘K’ values for proximity measures for SWaT dataset

| Proximity measure | Optimal ‘K’ value | Accuracy |
|---|---|---|
| Euclidean | 9 | 94.08 |
| Manhattan | 9 | 94.08 |
| Kulczynski | 9 | 94.08 |
| Chebyshev | 9 | 94.08 |
| Cosine | 9 | 94.08 |
| Soergel | 9 | 94.08 |
| Sorenson | 9 | 94.08 |
| Tanimoto | 9 | 94.08 |

## 5 Conclusion

Our study reveals that Kulczynski distance and Soergel distance are adequate with KNN to handle 2-class ransomware dataset with high classification accuracy. However, in case of multi-class data handling, although these two proximity measures have been found to assist winning performance in comparison to its other counterparts, the classification accuracies are relatively less. Interestingly, for SWaT dataset, among eight proximity measures, six measures such as Euclidean, Manhattan, Kulczynski, Cosine Similarity, Chebyshev, and Soergel distance are giving equal winning performances.

Out of all the computations performed, the Chebyshev distance for the bi- nary classification of Ransomware Dataset is least benefitted from the usage of Py-CUDA where GPU computation is only 40.86 times faster than the CPU computation and the Cosine Similarity for the classification of SWaT Dataset is highly benefitted from the usage of Py-CUDA where the GPU computation is 237.5 times faster than the CPU computation.

When dealing with the binary classification of the Ransomware Dataset using the KNN model, if accuracy is of high priority, then usage of Kulczynski or Soergel Distance is recommended. Similarly, when dealing with the multi-class classification of the Ransomware Dataset, if accuracy is of high priority, then usage of Kulczynski Distance is recommended. When dealing with the classification of the SWaT Dataset with high accuracy, usage of any of these six proximity measures is recommended. But if both the accuracy and the computational time are of high priority, then the usage of the Manhattan Distance is a better option to go with.

Also, we recommend $K$ values ranging from 2 to 9 for best possible accuracy for all the datasets used in the study. An exhaustive experimentation was also carried out for optimal feature selection based on some prominent feature selection algorithms [6, 7]. The performance of TUKNN with the optimal feature subset has been found significantly better than the present performance. However, due to lack of space, those results are not reported.

## References

1. Secure Water Treatment (SWaT) Dataset (2018). https://itrust.sutd.edu.sg/itrust-labsdatasets/datasetinfo/. Accessed 19 Dec 2018
2. Garcia, V., Debreuve, E., Barlaud, M.: Fast k nearest neighbor search using GPU. In: 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, pp. 1–6, June 2008. https://doi.org/10.1109/CVPRW.2008.4563100
3. Gogoi, P., Borah, B., Bhattacharyya, D.K.: Network anomaly identification using supervised classifier. Informatica **37**(1) (2013)
4. Han, J., Kamber, M., Pei, J.: Data Mining: Concepts and Techniques, 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco (2011)
5. Hand, D.J., Smyth, P., Mannila, H.: Principles of Data Mining. MIT Press, Cambridge (2001)
6. Hoque, N., Ahmed, H., Bhattacharyya, D., Kalita, J.: A fuzzy mutual information-based feature selection method for classification. Fuzzy Inf. Eng. **8**(3), 355–384 (2016)
7. Hoque, N., Singh, M., Bhattacharyya, D.K.: EFS-MI: an ensemble feature selection method for classification. Complex Intell. Syst. **4**(2), 105–118 (2018)

8. Liang, S., Liu, Y., Wang, C., Jian, L.: Design and evaluation of a parallel K-nearest neighbor algorithm on CUDA-enabled GPU. In: 2010 IEEE 2nd Symposium on Web Society, pp. 53–60, August 2010. https://doi.org/10.1109/SWS.2010.5607480
9. Lippert, A.: NVIDIA GPU architecture for general purpose computing (2009). Accessed 10 June 2018
10. Sgandurra, D., Muñoz-González, L., Mohsen, R., Lupu, E.C.: Automated dynamic analysis of ransomware: benefits, limitations and use for detection. arXiv preprint arXiv:1609.03020 (2016)
11. Tan, P.N.: Introduction to Data Mining. Pearson Education India (2018)