6.6 Discussion

The chapter introduces FRAMC, an ensemble method designed to effectively integrate individual feature rankings obtained from diverse base ranker algorithms using the Markov chain algorithm. Demonstrating its efficacy through benchmark datasets in moderate and high-dimensional spaces, FRAMC identifies an optimal number of features for both Windows and Android malware detection. While FRAMC exhibits promise, further investigation into alternative aggregation techniques holds the potential for even greater performance improvements. Enhancing the method's flexibility and robustness could involve exploring diverse ensemble tactics or fusion approaches. Future work could involve exploring the application of FRAMC as a cost-effective solution for IoT malware detection. By harnessing the efficiency and accuracy of FRAMC, it could be integrated into a holistic malware defense system, potentially with a novel classification method. This integration would not only enhance the capabilities of existing defense mechanisms but also provide a cost-effective solution for organizations facing evolving malware threats.

Chapter 7

Parallel k-Nearest Neighbors for Enhanced Malware Detection

7.1 Introduction

With the proliferation of data being generated, there is an urgent need of new technologies and architectures to make it possible to extract valuable information from it by capturing and analyzing processes. 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 large, different variety, structured or unstructured, which is referred to as Big data. Big Data is characterized by five particularly significant V's - Volume, Velocity, Variety, Veracity, and Value. The term Volume signifies the plethora of data produced from time to time by various 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. The term Veracity focuses on the quality and reliability of the data. With the increasing diversity and volume of data, ensuring the accuracy and trustworthiness of the information becomes crucial. The Value in the context of Big Data refers to the ultimate goal

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

of leveraging large and diverse datasets to extract meaningful insights

Apart from the mentioned characteristics above, two other key features are– incremental and dispersed nature. They are incremental in the sense that there is a 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 that set big data apart from traditional databases or data warehouses. The traditional data storage techniques are not adequate to store and analyze those huge volumes of data. In short, such 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 takes a 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 methods, to analyze big data. For example, many statistical methods that perform well for small data sizes 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 predicts what may happen in the future or gives the probability of what will happen in the future. We need to make use of such large data in order to make decisions in the future. However, traditional machine learning and statistical methods in sequential mode take much longer time in order to make predictions, especially, in case of intrusion data [88]. Despite their accuracy, these models are often found to be time-consuming, particularly when dealing with moderately sized datasets. This time delay could be a significant drawback in security scenarios where quick decision-making and response times are crucial.

7.1.1 Motivation

In various fields, the effectiveness of a predictive model is determined not only by its accuracy in predicting outcomes but also by its speed in generating those

predictions. This is particularly crucial in the domain of security applications, where there is a demand for a highly accurate predictive model capable of swiftly detecting any potential threat in near real-time. Unfortunately, conventional predictive models operating in sequential processing mode tend to be time-consuming, even when dealing with datasets of moderate size. This poses a challenge, as the need for quick and accurate responses in security scenarios requires more efficient predictive models that can perform effectively without significant delays.

The K-Nearest Neighbors [89] (K-NN) algorithm is widely recognized for its simplicity and effectiveness in classification tasks. However, one of its main drawbacks is its computational intensity, especially when dealing with large datasets. This is because K-NN requires calculating distances between the query point and all points in the dataset, which can be time-consuming. To address this issue, the development of a CUDA-accelerated K-NN algorithm becomes crucial, particularly in the context of malware and malware-based attacks. By leveraging the parallel processing capabilities of CUDA, the K-NN algorithm can be significantly accelerated, allowing for much faster computations. This enhancement in speed is essential for security applications, where the quick detection and response to malware threats are critical.

7.1.2 Contribution

The major contribution of this chapter is the development of a parallel version of the K-Nearest Neighbors (KNN) algorithm, referred to as TUKNN. This algorithm leverages parallel processing capabilities to enhance the speed and efficiency of KNN computations. An exhaustive experimental study is conducted on various proximity measures within the KNN framework, resulting in recommendations for the most effective measures to achieve better accuracy with the TUKNN algorithm. Additionally, the study identifies an optimal range for K values specifically for malware and malware-based attack datasets to ensure the best performance.

7.2 Background

7.2.1 Introduction to K-Nearest Neighbors (K-NN)

K-Nearest Neighbors (KNN) is a simple and intuitive machine-learning algorithm used for both classification and regression tasks. The fundamental idea behind KNN is to predict the class or value of a data point based on the majority class or average of its K nearest neighbors in the feature space. The term "K" represents the number of neighbors considered in the prediction process. To determine proximity, a distance measure is employed, commonly Euclidean distance, though other metrics like Manhattan or Minkowski distance can be used as well. The proximity measures quantify the similarity or dissimilarity between data points, with smaller distances indicating greater similarity. Essentially, the algorithm relies on the assumption that data points in close proximity in the feature space are likely to belong to the same class or share similar characteristics. The choice of K and the distance metric significantly impact the performance of the KNN algorithm, and practitioners often need to experiment with different values to optimize the model for specific datasets. KNN is particularly useful for scenarios where the decision boundaries are non-linear or difficult to define analytically. In the figure 7-1, the K-Nearest Neighbors (K-NN) algorithm is depicted within a 2-dimensional feature space. Blue circles represent data points, and the red circle indicates the query point for which a prediction is sought. The algorithm identifies the three nearest neighbors (shown as green circles) to the query point by calculating distances using a metric such as Euclidean distance. Dashed lines illustrate these distances. The K-NN algorithm then predicts the class or value of the query point based on the majority class or average value of the nearest neighbors, demonstrating how proximity in the feature space guides the prediction.

7.2.2 Proximity Measures

Proximity measures play a crucial role in quantifying the similarity or dissimilarity between data points in various domains such as data mining, machine learning, and information retrieval. Euclidean distance, a fundamental proximity measure, calculates the straight-line distance between two points in a multidimensional space. It is sensitive to the magnitude of differences along each axis. Conversely, Manhattan distance, also known as L1 norm, computes the distance as the sum of absolute differences along each axis, making it less sensitive to outliers. Kulczyn-

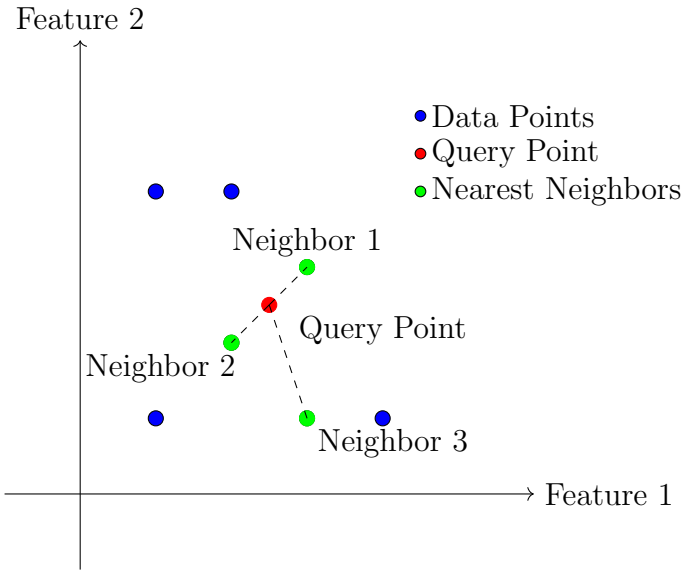

Figure 7-1: Illustration of the k-NN algorithm in 2D

ski distance assesses the dissimilarity between two sets based on the proportion of common elements relative to the average size of the sets. Cosine similarity, widely used in text analysis, measures the cosine of the angle between two vectors, representing the direction of similarity irrespective of magnitude. Chebyshev distance determines the greatest absolute difference along any coordinate axis, making it suitable for scenarios where only the maximum deviation matters. Soergel distance and Sorensen measure dissimilarity in binary data, with Soergel emphasizing larger differences and Sorensen focusing on shared binary features. Tanimoto distance, often used in set-based comparisons, assesses dissimilarity by considering the ratio of the intersection to the union of sets. These proximity measures provide diverse tools for capturing different aspects of similarity or dissimilarity, allowing researchers and practitioners to choose the most appropriate method based on the characteristics of their data and the context of their analysis.

7.2.3 Introduction to CUDA

CUDA, which stands for Compute Unified Device Architecture, is a parallel computing platform and programming model developed by NVIDIA [90]. It enables developers to use NVIDIA graphics processing units (GPUs) for general-purpose processing, going beyond their traditional role in rendering graphics. The architecture of CUDA is designed to harness the parallel processing capabilities of GPUs, allowing for significant acceleration of computation-intensive tasks.

At the core of CUDA's architecture is the Streaming Multiprocessor (SM), which is a fundamental building block of NVIDIA GPUs. SMs consist of multiple CUDA

cores, which are individual processing units capable of executing parallel threads. These threads are organized into blocks, and multiple blocks are grouped into a grid. This hierarchical structure facilitates the parallel execution of tasks, with each thread responsible for a specific portion of the computation. CUDA utilizes a Single Instruction, Multiple Thread (SIMT) model, where threads within a block execute the same instruction, but each thread operates on different data.

The memory hierarchy is a critical aspect of CUDA architecture. It includes global memory, shared memory, and constant memory. Global memory is accessible by all threads but has higher latency. Shared memory is a fast, on-chip memory that threads within a block can share, enabling efficient communication and collaboration. Constant memory is used for storing constant data that remains unchanged throughout the execution of a kernel. Registers memory is used for storing temporary variables for threads.

CUDA programming involves writing kernels, which are functions executed on the GPU. These kernels are written in CUDA C, a parallel computing extension of the C programming language. Developers explicitly define which parts of the code should be executed on the CPU and GPU, and data transfer between the CPU and GPU is managed through explicit memory management functions.

The CUDA processing flow involves a series of steps that leverage the parallel processing capabilities of GPUs to accelerate computations. The process typically begins on the Host CPU, where data is initialized and prepared for processing. This initial data may represent tasks, algorithms, or calculations that can benefit from parallel execution. Once the data is ready, it is transferred from the Host Memory to the GPU Memory. This data transfer is a crucial step in utilizing the parallel architecture of the GPU, enabling efficient parallel processing on a large scale.

On the GPU, the actual computation takes place within CUDA Kernels. Kernels are specialized functions written in CUDA C that are executed in parallel by multiple threads on the GPU. Each thread processes a specific portion of the data, allowing for concurrent execution and substantial speedup compared to sequential processing on the CPU. The GPU Memory plays a crucial role during kernel execution, storing not only the input data but also intermediate results generated by parallel threads.

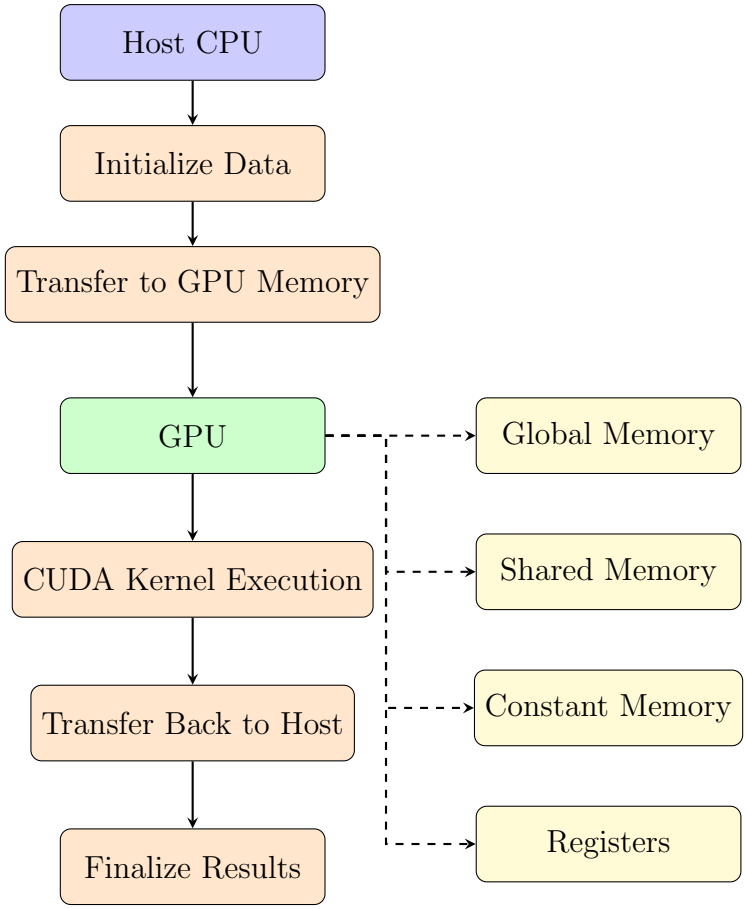

Figure 7-2: Illustration of the CUDA concept and workflow

Figure 7-2 illustrates the CUDA concept and workflow, starting with data initialization on the Host CPU. The data is then transferred to the GPU memory, which includes Global Memory, Shared Memory, Constant Memory, and Registers. The core computational process occurs in the "CUDA Kernel Execution" step, where multiple threads process data in parallel. After the computation, the results are transferred back to the Host CPU for finalization. This workflow highlights how CUDA leverages GPU parallel processing and various memory types to perform efficient and high-speed computations.

7.3 Problem Statement

The objective of this research is to efficiently classify a query point q using the k-nearest neighbors (k-NN) approach, leveraging the parallel processing capabilities of CUDA through the TUKNN algorithm. Given a dataset D with N data points, each represented as a vector x_i in a d-dimensional feature space, the TUKNN algorithm involves the following steps:

1. **Distance Computation**: Compute the distance between the query point q and each data point x_i in parallel. For instance, using the Euclidean distance:

$$\text{distance}(q, x_i) = \sqrt{\sum_{j=1}^{d} (q_j - x_{ij})^2}$$

2. **Nearest Neighbor Selection**: Identify the k-nearest neighbors with the smallest distances to the query point:

$$\text{neighbors}(q, D, k) = \text{argmin}_{i=1}^{N} \left(\text{distance}(q, x_i)\right)$$

3. **Majority Voting**: Determine the majority class among the k-nearest neighbors to classify or predict the label of the query point:

$$\text{prediction}(q, D, k) = \text{majority\_class}\left(\text{labels}\left(\text{neighbors}(q, D, k)\right)\right)$$

where $\text{labels}(N_{\text{neighbors}})$ returns the class labels of the neighbors, and $\text{majority\_class}(L)$ returns the class with the highest count in list L.

7.4 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 figure 7-3. 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.

7.4.1 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

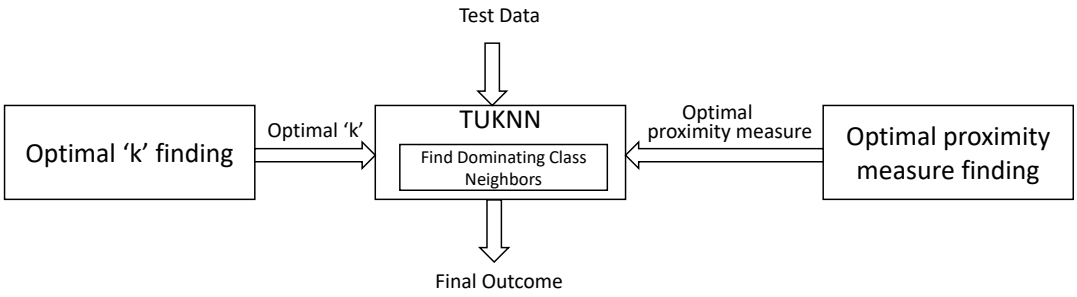

Figure 7-3: Framework of the proposed work

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 7.1 shows the distance measures and their mathematical expressions used in our work.

- *Euclidean Distance*: It is the most common distance measure that is used in the KNN algorithm. The Euclidean distance between the two points measures the length of a segment connecting between two points. The formula for Euclidean distance is given below.

$$D_{xy} = \sqrt{(\sum_{k=1}^{m}(x_{ik} - y_{jk})^2)} \tag{7.1}$$

- *Manhattan Distance*: In Manhattan distance the distance between two points is the sum of the absolute difference of their cartesian coordinates.The distance is calculated using the formula given below.

$$D_{Manhattan}(x, y) = |x_i - y_i| \tag{7.2}$$

- *Kulczynski Distance*: In Kulczynski distance measure, the distance between two points is the ratio of the sum of the absolute difference of their Cartesian coordinates and the sum of the minimum of their Cartesian coordinates. It is defined as follows.

$$D_{kulczynski}(x, y) = \frac{(\sum |x_i - y_i|)}{(\sum max(x_i, y_i))} \tag{7.3}$$

- *Chebyshev Distance*:It is also called as maximum value distance. It calculates the absolute magnitudes of the differences between coordinates of a pair of objects. The Chebyshev distance between two points or vector x and y with standard coordinate x_i and y_i is computed as follows.

$$D_{Chebyshev}(x, y) = max_i(|x_i - y_i|) \tag{7.4}$$

Table 7.1: Distance measures and their mathematical expressions [1] [2]

Measure & References	Math Expression
Euclidean [91]	$D_{xy} = \sqrt{(\sum_{i=1}^{m}(x_i - y_i)^2)}$
Manhattan [92]	$D_{Manhattan}(x, y) = \lvert x_i - y_i \rvert$
Kulczynski [93]	$D_{kulczynski}(x, y) = \frac{(\sum \lvert x_i - y_i \rvert)}{(\sum max(x_i, y_i))}$
Chebyshev [94]	$D_{Chebyshev}(x, y) = max_i(\lvert x_i - y_i \rvert)$
Cosine [92]	$Sim(A, B) = cos(\theta) = \frac{A.B}{\lvert\lvert A \rvert\rvert \lvert\lvert B \rvert\rvert}$
Sorgel [95]	$D_{sg} = \frac{\sum_{i=1}^{d} \lvert P_i - Q_i \rvert}{\sum_{i=1}^{d} min(P_i, Q_i)}$
Sorenson [96]	$D_{soresnosn} = \frac{2\lvert x.y \rvert}{\lvert x \rvert^2 + \lvert y \rvert^2}$
Tanimoto [97]	$D_{tanimoto} = \frac{x.y}{\lvert x \rvert * \lvert x \rvert + \lvert y \rvert * \lvert y \rvert - x.y}$

- *Cosine Similarity*:It is the measure of calculating the difference between the angle of the two vectors. The cosine similarity is calculated using the formula given below.

$$Sim(A, B) = cos(\theta) = \frac{A.B}{||A||||B||} \tag{7.5}$$

- *Soergel Distance*: The Soergel Distance measure is given by the formula below.

$$d_{sg} = \frac{\sum_{i=1}^{d} |P_i - Q_i|}{\sum_{i=1}^{d} min(P_i, Q_i)} \tag{7.6}$$

- *Sorenson Distance*: The sorenson distance between any two vector can be obtained by the following formula.

$$d_{soresnosn} = \frac{2|x.y|}{|x|^2 + |y|^2} \tag{7.7}$$

- *Tanimoto Distance*: The tanimoto distance between any two vector can be obtained by the following formula.

$$d_{tanimoto} = \frac{x.y}{|x| * |x| + |y| * |y| - x.y} \tag{7.8}$$

Further, exhaustive experimentation was carried out on a large number of datasets by varying the K values to identify an optimal range of K values for the best possible performance of TUKNN. Additionally, both the sequential and parallel versions of the KNN algorithm were presented.

7.4.2 Sequential KNN algorithm

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

[2] Split the dataset into a test set and training set using 5-fold cross-validation.

[3] For every test instance in the test set perform 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 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 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 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.

7.4.3 TUKNN Algorithm

The algorithm for parallel KNN implementation is stated below.

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

[2] Split the dataset into a test set and training set using 5-fold cross-validation.

[3] For every n instances(2500) in the test set, perform 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 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 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.

7.4.4 Complexity Analysis

The time complexity of k-Nearest Neighbors (k-NN) is $O(N \cdot D)$, where N is the number of data points and D is the dimensionality of the data. This complexity arises because, during prediction, the algorithm must compute the distances between the query point and all other data points.

Although the time complexity of TUKNN in terms of big-O notation remains the same ($O(N \cdot D)$), the actual execution time is significantly reduced due to parallelization. By leveraging parallel processing capabilities, TUKNN distributes the distance calculations across multiple threads on the GPU, leading to a substantial decrease in computation time.

7.5 Implementation and results

For the parallel KNN, we have computed all of the distances between a set of test instances and all the training instances simultaneously. Hence, all the distances are computed at once, parallelly. To calculate the distance between the test instances and all training instances in parallel, we have used many cores of the GPU platform and developed 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 have implemented 'd', the distance computation on the GPU platform to perform parallel computation, which has resulted in considerable improvement in the KNN performance.

The graphics card used for this project was an NVIDIA Tesla K40c GPU Accelerator, which has 12GB of memory. With this memory capacity, the GPU was able to compute the distances between 2500 test instances and all the training instances in the dataset simultaneously.

7.5.1 Dataset Used

We performed our experimentation on the following datasets.

1. Ransomware Dataset: For our experiment, we use a dataset from Sgandurra et al. [98]. The dataset contains 582 samples of ransomware with 11 variants and 942 samples of benign programs. The dataset has 30,962 attributes which represent all instances both goodware and ransomware present in the dataset. A detailed description of the dataset is given in the Table 7.2.

Table 7.2: 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

2. SWAT Dataset [30]: SWaT represents a scaled down version of a real-world industrial water treatment plant producing 5 gallons per minute of water filtered via membrane based ultrafiltration and reverse osmosis units. The main purpose of the dataset, carried out by the research team (Sridhar Adepu and team) was to design secure and safe CPS (Cyber Physical

System). SWaT has six main processes corresponding to the physical and control components of the water treatment facility. In total, 946,722 samples comprising of 51 attributes were collected over 11 days. The dataset consists of two labels- "Attack" and "Normal", where all the different types of attacks are merged into a single class under label - "Attack".

3. UCI datasets: A total of 20 datasets is also used in our work. The list of datasets used for this purpose given in the table 7.3.

Table 7.3: 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

7.5.2 Results and Observation

In our proposed methodology for KNN, the value of K is determined after twenty datasets from UCI Machine Learning repository were tested. This testing reduces

the overhead of calculating the best K value for highest accuracy and makes our model faster. As we can see form the table 7.4, in the majority cases (15 out of 20) the results show optimal K values within the range of 2-10.

Table 7.4: Value of K for which the maximum accuracy is achieved for the 20 UCI ML Repository Dataset

S.I.	Dataset Name	Instances	Features	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%

7.5.2.1 Results for Binary Classification on Ransomware Dataset

We first performed binary classification on the ransomware dataset. In the binary classification 'Goodware' is given the class label 0 and rest all the family member of the dataset in the table is labeled as 1.

1. Result for Euclidean Distance

Table 7.5: Results of Euclidean Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.924590164	0.898360656	0.901639344	0.947540984	0.894389439	0.913304117
3	0.931147541	0.91147541	0.921311475	0.950819672	0.917491749	0.926449169
4	0.927868852	0.88852459	0.898360656	0.93442623	0.900990099	0.910034085
5	0.931147541	0.891803279	0.895081967	0.947540984	0.907590759	0.914632906
6	0.924590164	0.88852459	0.878688525	0.931147541	0.90429043	0.90544825
7	0.927868852	0.88852459	0.881967213	0.940983607	0.920792079	0.912027268
8	0.914754098	0.878688525	0.878688525	0.931147541	0.904290429	0.901513824
9	0.914754098	0.875409836	0.885245902	0.921311475	0.897689769	0.898882216
10	0.924590164	0.875409836	0.868852459	0.904918032	0.877887789	0.890331656

Table 7.6: Time Comparison betwenn CPU and GPU

CPU time	GPU time
1hr 21mins 20 sec	73.606 seconds

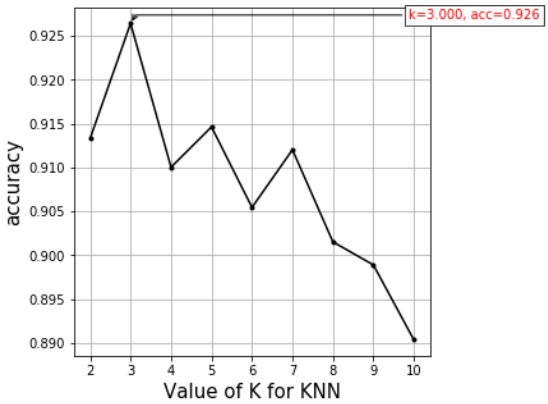

Figure 7-4: K vs accuracy graph for Euclidean Distance

2. Result for Manhattan Distance

Table 7.7: Result for Manhattan Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.924590164	0.898360656	0.901639344	0.947540984	0.894389438	0.913304117
3	0.931147541	0.91147541	0.921311475	0.950819672	0.917491749	0.926449169
4	0.927868852	0.88852459	0.898360655	0.934426229	0.900990099	0.910034085
5	0.93114754	0.891803278	0.895081967	0.947540983	0.907590759	0.914632905
6	0.924590163	0.88852459	0.878688524	0.93114754	0.904290429	0.905448249
7	0.927868852	0.88852459	0.881967213	0.940983606	0.920792079	0.912027268
8	0.914754098	0.878688524	0.878688524	0.93114754	0.904290429	0.901513823
9	0.914754098	0.875409836	0.885245901	0.921311475	0.897689768	0.898882215
10	0.924590163	0.875409836	0.868852459	0.904918032	0.877887788	0.890331655

Table 7.8: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
58mins 30 sec	72.994 seconds

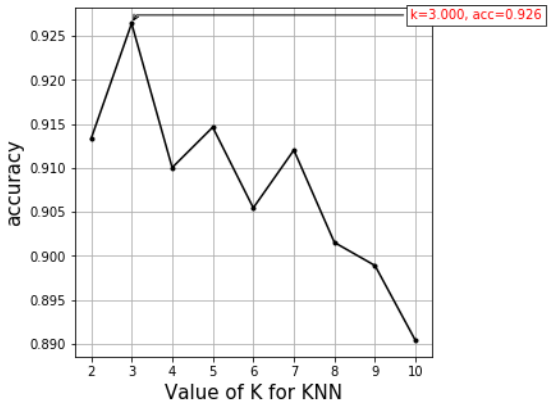

Figure 7-5: K vs accuracy graph for Manhattan Distance

3. Result for Kulczynski distance

Table 7.9: Result for Kulczynski distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.960655737	0.93114754	0.96065573	0.96065573	0.94389438	0.95140182
3	0.94754098	0.92786885	0.96393442	0.97377049	0.95049504	0.95272195
4	0.93770491	0.93114754	0.96393442	0.96393442	0.94719471	0.9487832
5	0.93770491	0.92459016	0.95081967	0.95409836	0.94059405	0.94156143
6	0.93770491	0.92131147	0.95081967	0.94754098	0.95379537	0.94223448
7	0.93770491	0.9114754	0.9409836	0.95081967	0.96039603	0.94027592
8	0.93442622	0.91803278	0.93770491	0.9409836	0.94719471	0.93566844
9	0.93442622	0.90819672	0.93114754	0.9409836	0.95379537	0.93370989
10	0.93442622	0.91803278	0.92786885	0.94426229	0.95049504	0.93501703

Table 7.10: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
6664.25 seconds	76.176 seconds

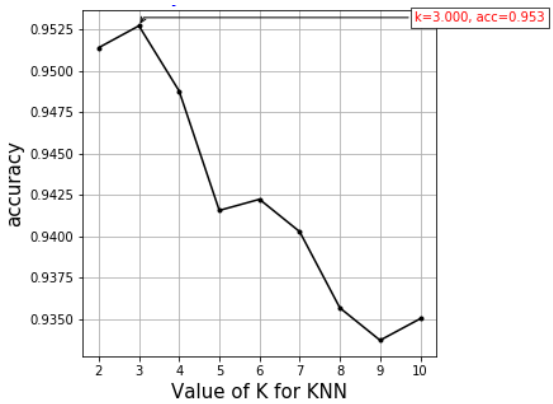

Figure 7-6: K vs accuracy graph for Kulczynski Distance

4. Results for Chebyshev Distance

Table 7.11: Results for Chebyshev Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.701639344	0.711475409	0.685245901	0.695081967	0.689768976	0.696642319
3	0.675409836	0.698360655	0.662295081	0.675409836	0.663366336	0.674968348
4	0.675409836	0.698360655	0.662295081	0.675409836	0.663366336	0.674968348
5	0.668852459	0.675409836	0.652459016	0.66557377	0.646864686	0.661831953
6	0.668852459	0.675409836	0.652459016	0.66557377	0.646864686	0.661831953
7	0.659016393	0.672131147	0.65245901	0.655737704	0.636963696	0.65526159
8	0.704918032	0.672131147	0.65245901	0.655737704	0.636963696	0.664441917
9	0.675409836	0.66557377	0.652459016	0.655737704	0.636963696	0.657228804
10	0.675409836	0.675409836	0.652459016	0.655737704	0.646864686	0.661176215

Table 7.12: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
51mins 5 seconds	74.55100226402283 seconds

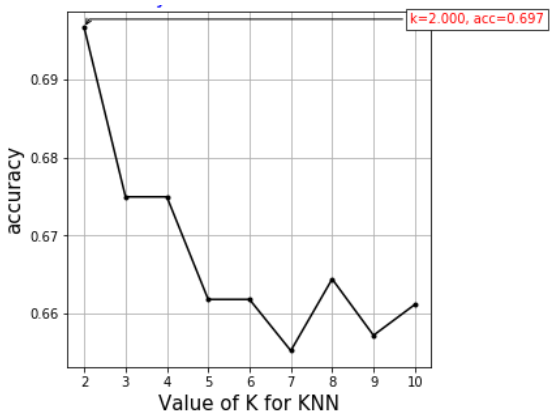

Figure 7-7: K vs accuracy graph for Chebyshev Distance

5. Result for Cosine Similarity

Table 7.13: Result for Cosine Similarity

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.547540983	0.6	0.534426229	0.540983606	0.528052805	0.550200724
3	0.573770491	0.485245901	0.563934426	0.626229508	0.594059405	0.568647946
4	0.498360655	0.491803278	0.478688524	0.629508196	0.561056105	0.531883352
5	0.593442622	0.649180327	0.616393442	0.622950819	0.590759075	0.614545257
6	0.593442622	0.645901639	0.616393442	0.619672131	0.56435643	0.607953252
7	0.606557377	0.629508196	0.629508196	0.626229508	0.607260726	0.6198128
8	0.603278688	0.629508196	0.622950819	0.622950819	0.603960396	0.616529783
9	0.606557377	0.626229508	0.626229508	0.626229508	0.607260726	0.618501325
10	0.606557377	0.622950819	0.626229508	0.626229508	0.607260726	0.617845587

Table 7.14: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
1hr 56mins 40 seconds	74.05940866470337 seconds

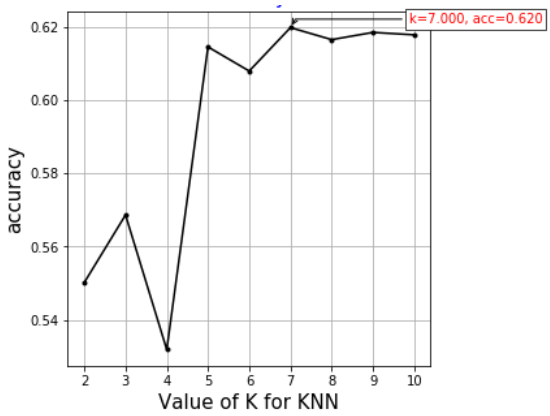

Figure 7-8: K vs accuracy graph for Cosine similarity measure

6. Result for Soergel Distance

Table 7.15: Result for Soergel Distance

K Value	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.96065573	0.93114754	0.96065573	0.96065573	0.94389438	0.95140182
3	0.94754098	0.92786885	0.96393442	0.97377049	0.95049504	0.95272195
4	0.93770491	0.93114754	0.96393442	0.96393442	0.94719471	0.9487832
5	0.93770491	0.92459016	0.95081967	0.95409836	0.94059405	0.94156143
6	0.93770491	0.92131147	0.95081967	0.94754098	0.95379537	0.94223448
7	0.93770491	0.9114754	0.9409836	0.95081967	0.96039603	0.94027592
8	0.93442622	0.91803278	0.93770491	0.9409836	0.94719471	0.93566844
9	0.93442622	0.90819672	0.93114754	0.9409836	0.95379537	0.93370989
10	0.93442622	0.91803278	0.92786885	0.94426229	0.95049504	0.93501703

Table 7.16: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
6633.75 seconds	73.79 seconds

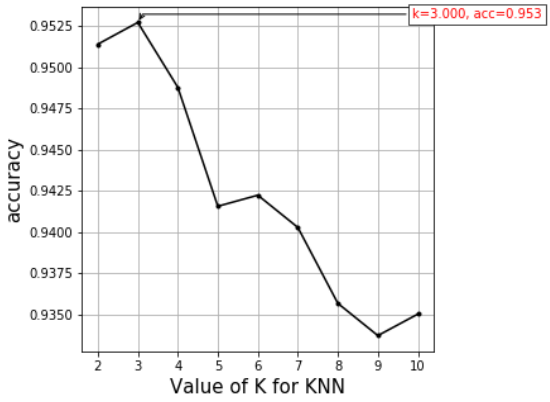

Figure 7-9: K vs accuracy graph for Sorgel distance measure

7. Results for Sorenson Distance

Table 7.17: Results for Sorenson Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.96065573	0.93114754	0.96065573	0.96065573	0.94389438	0.95140182
3	0.94754098	0.92786885	0.96393442	0.97377049	0.95049504	0.95272195
4	0.93770491	0.93114754	0.96393442	0.96393442	0.94719471	0.9487832
5	0.93770491	0.92459016	0.95081967	0.95409836	0.94059405	0.94156143
6	0.93770491	0.92131147	0.95081967	0.94754098	0.95379537	0.94223448
7	0.93770491	0.9114754	0.9409836	0.95081967	0.96039603	0.94027592
8	0.93442622	0.91803278	0.93770491	0.9409836	0.94719471	0.93566844
9	0.93442622	0.90819672	0.93114754	0.9409836	0.95379537	0.93370989
10	0.93442622	0.91803278	0.92786885	0.94426229	0.95049504	0.93501703

Table 7.18: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
5901.75 seconds	151.07 seconds

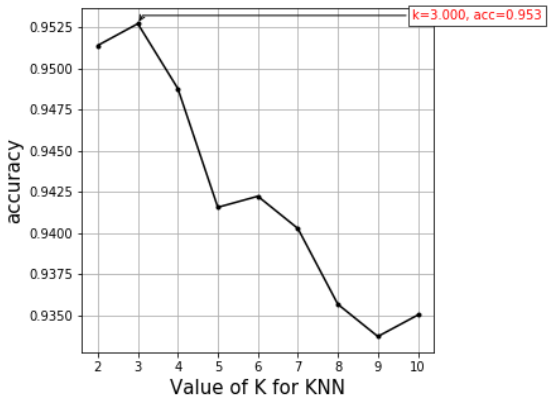

Figure 7-10: K vs accuracy graph for soreson distance measure

8. Results for Tanimoto Distance

Table 7.19: Results for Tanimoto Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.96065573	0.93114754	0.96065573	0.96065573	0.94389438	0.95140182
3	0.94754098	0.92786885	0.96393442	0.97377049	0.95049504	0.95272195
4	0.9377049	0.93114754	0.96393442	0.96393442	0.94719471	0.94878319
5	0.9377049	0.92459016	0.95081967	0.95409836	0.94059405	0.94156142
6	0.9377049	0.92131147	0.95081967	0.94754098	0.95379537	0.94223447
7	0.9377049	0.9114754	0.9409836	0.95081967	0.96039603	0.94027592
8	0.93442622	0.91803278	0.93770491	0.9409836	0.94719471	0.93566844
9	0.93442622	0.90819672	0.93114754	0.9409836	0.95379537	0.93370989
10	0.93442622	0.91803278	0.92786885	0.94426229	0.95049504	0.93501703

Table 7.20: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
4925.75 seconds	138.11 seconds

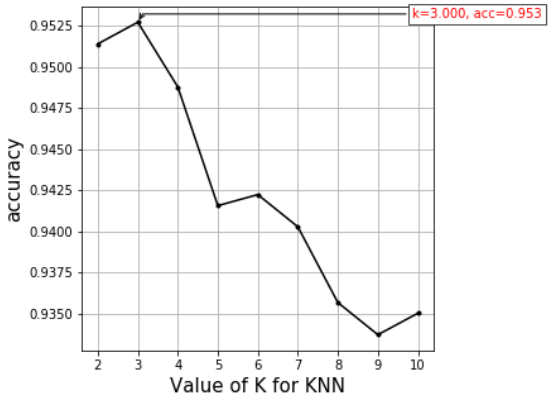

Figure 7-11: K vs accuracy graph for tanimoto distance measure

7.5.2.2 Result for Multi-Class Classification on Ransomware Dataset

1. Result for Euclidean Distance

Table 7.21: Result for Euclidean Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.8262295	0.77377049	0.79344262	0.81311475	0.75577557	0.79246658
3	0.79672131	0.76721311	0.77377049	0.81311475	0.75247524	0.78065898
4	0.80983606	0.74754098	0.76065573	0.81639344	0.73267326	0.77341989
5	0.79672131	0.757377049	0.7409836	0.8	0.73927392	0.76687117
6	0.79016393	0.74754098	0.73442622	0.80327868	0.74257425	0.76359681
7	0.78360655	0.74754098	0.73442622	0.79344262	0.7260726	0.75701779
8	0.79016393	0.75081967	0.73770491	0.78688524	0.73267326	0.7596494
9	0.79016393	0.74754098	0.75081967	0.79344262	0.72937293	0.76226802
10	0.77704918	0.7409836	0.75081967	0.79672131	0.72277227	0.7576692

Table 7.22: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
1hr 34mins 5sec	74.730847120285 seconds

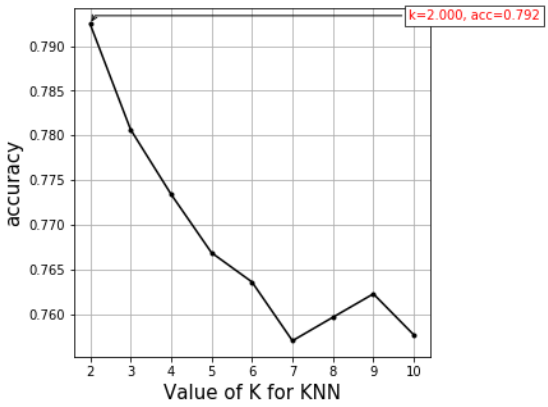

Figure 7-12: K vs accuracy graph for Euclidean distance measure

2. Result for Manhattan Distance

Table 7.23: Result for Manhattan Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.8262295	0.77377049	0.79344262	0.81311475	0.75577557	0.79246658
3	0.79672131	0.76721311	0.77377049	0.81311475	0.75247524	0.78065898
4	0.80983606	0.74754098	0.76065573	0.81639344	0.73267326	0.77341989
5	0.79672131	0.757377049	0.7409836	0.8	0.73927392	0.76687117
6	0.79016393	0.74754098	0.73442622	0.80327868	0.74257425	0.76359681
7	0.78360655	0.74754098	0.73442622	0.79344262	0.7260726	0.75701779
8	0.79016393	0.75081967	0.73770491	0.78688524	0.73267326	0.7596494
9	0.79016393	0.74754098	0.75081967	0.79344262	0.72937293	0.76226802
10	0.77704918	0.7409836	0.75081967	0.79672131	0.72277227	0.7576692

Table 7.24: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
1hr 11mins 10sec	80.1867 seconds

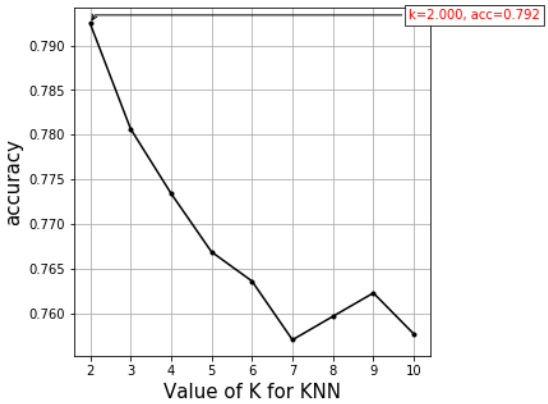

Figure 7-13: K vs accuracy graph for Manhattan distance measure

3. Result for Kulczynski distance

Table 7.25: Result for Kulczynski distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.8262295	0.79344262	0.81311475	0.84262295	0.7722772	0.8095374
3	0.82295081	0.78032786	0.8	0.83606557	0.7722772	0.8232428
4	0.80983606	0.76393442	0.78688524	0.82295081	0.77227722	0.7911767
5	0.79344262	0.78688524	0.77377049	0.82295081	0.8229508	0.7999999
6	0.80327868	0.79016393	0.77704918	0.80983606	0.76567656	0.7892008
7	0.79672131	0.78360655	0.77377049	0.81967213	0.76567656	0.7878894
8	0.79016393	0.78360655	0.77377049	0.81639344	0.76567656	0.78592219
9	0.79344262	0.78360655	0.77377049	0.81639344	0.76567656	0.7865779
10	0.78688524	0.78360655	0.77377049	0.81639344	0.76567656	0.78526645

Table 7.26: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
7777.5 Seconds	77.054 Seconds

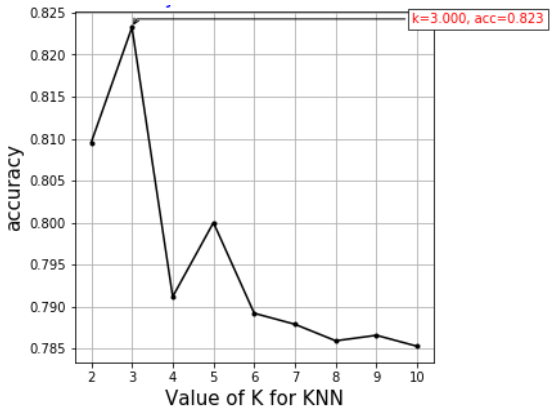

Figure 7-14: K vs accuracy graph for Kulczynski distance measure

4. Result for Chebyshev Distance

Table 7.27: Result for Chebyshev Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.67213114	0.69180327	0.65901639	0.67540983	0.65346534	0.67036519
3	0.67213114	0.69180327	0.65901639	0.66885245	0.65346534	0.66905371
4	0.66885245	0.67540983	0.64918032	0.66557377	0.64026402	0.65985607
5	0.66885245	0.67540983	0.64918032	0.66557377	0.64026402	0.65985607
6	0.65901639	0.67213114	0.64918032	0.6557377	0.63366336	0.65394578
7	0.65901639	0.67213114	0.64918032	0.6557377	0.63366336	0.65394578
8	0.64590163	0.66557377	0.64918032	0.6557377	0.63366336	0.65001135
9	0.64590163	0.66557377	0.64918032	0.6557377	0.63366336	0.65001135
10	0.63934426	0.66229508	0.64918032	0.64590163	0.63036303	0.64541686

Table 7.28: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
1hr 3mins 35sec	75.6112 Seconds

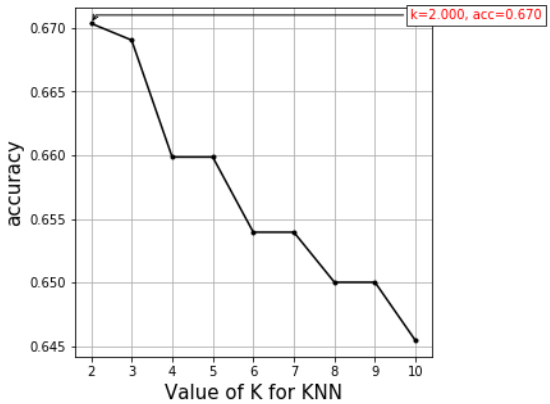

Figure 7-15: K vs accuracy graph for Chebyshev distance measure

5. Result for Cosine Similarity

Table 7.29: Result for Cosine Similarity

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.58688524	0.4262295	0.59344262	0.6262295	0.60396039	0.56734945
3	0.60655737	0.62295081	0.6262295	0.62295081	0.60726072	0.61718984
4	0.60655737	0.62295081	0.6262295	0.62295081	0.60726072	0.61718984
5	0.60655737	0.62295081	0.6262295	0.62295081	0.5940594	0.61454957
6	0.60655737	0.62295081	0.6262295	0.62295081	0.60726072	0.61718984
7	0.60655737	0.62295081	0.6262295	0.62295081	0.60726072	0.61718984
8	0.60655737	0.6262295	0.6262295	0.62295081	0.60726072	0.61784558
9	0.60655737	0.6262295	0.6262295	0.62295081	0.60726072	0.61784558
10	0.60655737	0.6262295	0.6262295	0.62295081	0.60726072	0.61784558

Table 7.30: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
2hr 7mins 50sec	75.6016 sec

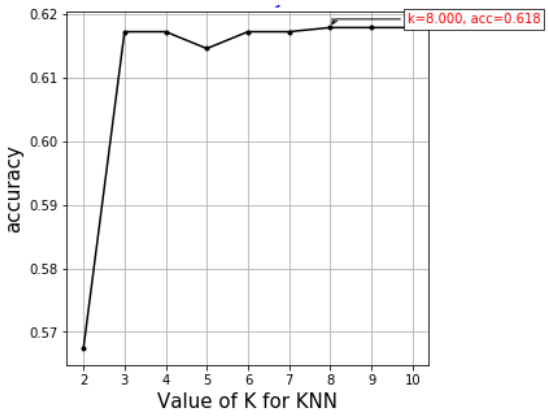

Figure 7-16: K vs accuracy graph for Cosine similarity measure

6. Result for Soergel Distance

Table 7.31: Result for Soergel Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.8262295	0.79344262	0.81311475	0.84262295	0.77227722	0.8095374
3	0.82295081	0.78032786	0.8	0.83606557	0.77227722	0.80232429
4	0.80983606	0.76393442	0.78688524	0.82295081	0.77227722	0.79117675
5	0.79344262	0.78688524	0.77377049	0.82295081	0.76897689	0.78920521
6	0.80327868	0.79016393	0.77704918	0.80983606	0.76567656	0.78920088
7	0.79672131	0.78360655	0.77377049	0.81967213	0.76567656	0.7878894
8	0.79016393	0.78360655	0.77377049	0.81639344	0.76567656	0.78592219
9	0.79344262	0.78360655	0.77377049	0.81639344	0.76567656	0.78657793
10	0.78688524	0.78360655	0.77377049	0.81639344	0.76567656	0.78526645

Table 7.32: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
7701.25 seconds	73.64 seconds

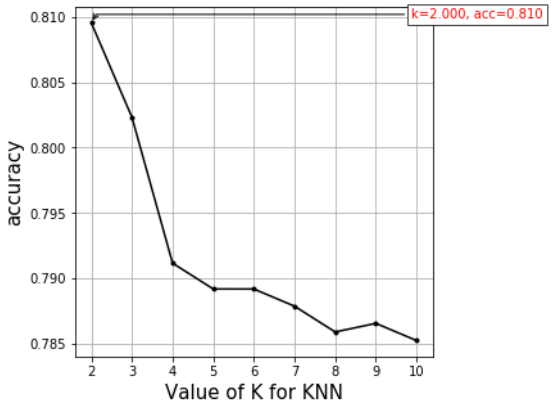

Figure 7-17: K vs accuracy graph for Sorgel distance similarity measure

7. Results for Sorenson Distance

Table 7.33: Results for Sorenson Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.63606557	0.64918032	0.63934426	0.64590163	0.62376237	0.63885083
3	0.63278688	0.64590163	0.63934426	0.64590163	0.62376237	0.63753935
4	0.63278688	0.64590163	0.63934426	0.64918032	0.62376237	0.63819509
5	0.63278688	0.64918032	0.63934426	0.64918032	0.62376237	0.63885083
6	0.63278688	0.64918032	0.63934426	0.64918032	0.62376237	0.63885083
7	0.63278688	0.64918032	0.63934426	0.64918032	0.62376237	0.63885083
8	0.63278688	0.64590163	0.63934426	0.64918032	0.62376237	0.63819509
9	0.63278688	0.64590163	0.63934426	0.64918032	0.62376237	0.63819509
10	0.63278688	0.64918032	0.63934426	0.64918032	0.62376237	0.63885083

Table 7.34: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
6100 seconds	142.91 seconds

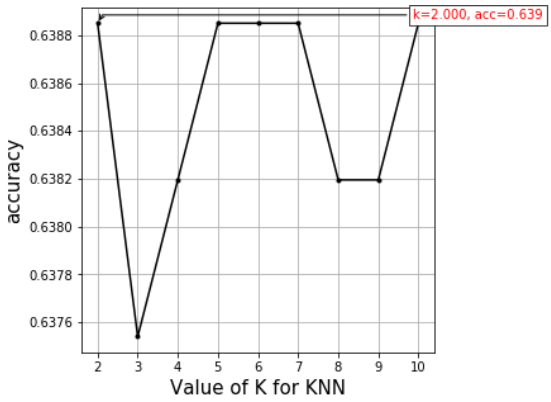

Figure 7-18: K vs accuracy graph for Sorenson distance similarity measure

8. Results for Tanimoto Distance

Table 7.35: Results for Tanimoto Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.63606557	0.64918032	0.63934426	0.64590163	0.62376237	0.63885083
3	0.63278688	0.64590163	0.63934426	0.64590163	0.62376237	0.63753935
4	0.63278688	0.64590163	0.63934426	0.64918032	0.62376237	0.63819509
5	0.63278688	0.64918032	0.63934426	0.64918032	0.62376237	0.63885083
6	0.63278688	0.64918032	0.63934426	0.64918032	0.62376237	0.63885083
7	0.63278688	0.64918032	0.63934426	0.64918032	0.62376237	0.63885083
8	0.63278688	0.64590163	0.63934426	0.64918032	0.62376237	0.63819509
9	0.63278688	0.64590163	0.63934426	0.64918032	0.62376237	0.63819509
10	0.63278688	0.64918032	0.63934426	0.64918032	0.62376237	0.63885083

Table 7.36: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
5337.5 seconds	137.90 seconds

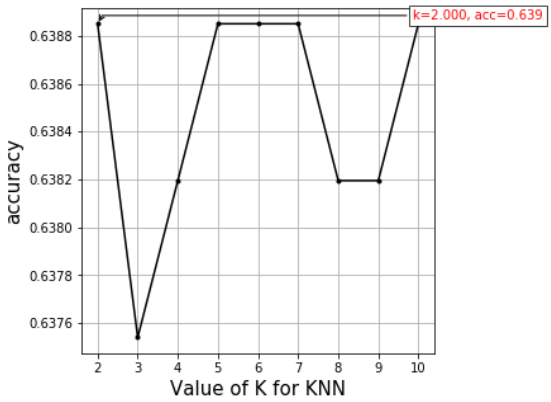

Figure 7-19: K vs accuracy graph for Tanimoto distance similarity measure

7.5.2.3 Results for SWaT Dataset

1. Result for Euclidean Distance

Table 7.37: Result for Euclidean Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.84034672	0.8387451	0.84121787	0.84009241	0.84206852	0.8404941
3	0.93216386	0.93148209	0.931639	0.93131976	0.93346608	0.9320141
4	0.92373913	0.92330085	0.9237283	0.92333331	0.92553893	0.9239281
5	0.93940902	0.93911683	0.93866232	0.93831061	0.94027314	0.9391543
6	0.93805089	0.9373583	0.93693084	0.93695789	0.93874724	0.937609
7	0.94077256	0.94055071	0.9400475	0.93975532	0.94175035	0.9405752
8	0.94050202	0.94018819	0.93973367	0.93945231	0.94139322	0.9402538
9	0.94105393	0.94083749	0.94030722	0.9400475	0.94205337	0.9408599
10	0.94102687	0.94074551	0.94025853	0.93998798	0.94197762	0.9407993

Table 7.38: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
59days 4hrs 25mins	9hrs 47mins 55sec

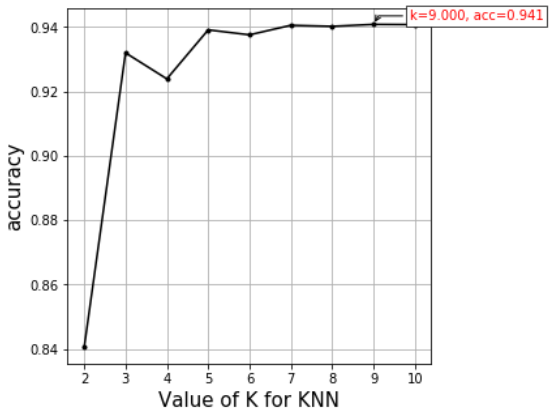

Figure 7-20: K vs accuracy graph for Euclidean distance similarity measure

2. Result for Manhattan Distance

Table 7.39: Result for Manhattan Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.84031426	0.83861524	0.84106637	0.84002207	0.8415653	0.84031664
3	0.93215845	0.93180674	0.93192037	0.93113038	0.93329834	0.93206285
4	0.92380947	0.9234253	0.923793239	0.92336036	0.92513311	0.92390429
5	0.93929539	0.93920341	0.93872725	0.93837554	0.94026773	0.93917386
6	0.93793726	0.9374611	0.93702282	0.93690378	0.93872559	0.93761011
7	0.94072927	0.94056695	0.94001504	0.93970121	0.94169624	0.94054174
8	0.94046414	0.94019901	0.93971203	0.9393927	0.94136617	0.94022681
9	0.9410431	0.94085372	0.94026394	0.94000422	0.94203173	0.94083934
10	0.94100523	0.94076174	0.94017736	0.93995011	0.94195056	0.940769

Table 7.40: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
51days 20mins	9hrs 29mins 50sec

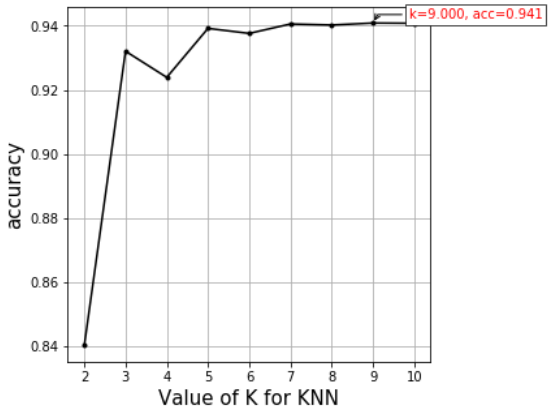

Figure 7-21: K vs accuracy graph for Manhattan distance similarity measure

3. Result for Kulczynski Distance

Table 7.41: Result for Kulczynski Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.84030885	0.83862065	0.84106637	0.84003289	0.8415653	0.84031881
3	0.93215845	0.93180133	0.93192037	0.93113579	0.93329834	0.93206285
4	0.92381488	0.9234253	0.92379865	0.92335495	0.92513311	0.92390537
5	0.9392953	0.93920341	0.93872725	0.93837554	0.94026773	0.93917384
6	0.93793185	0.9374611	0.93702282	0.93690378	0.938731	0.93761011
7	0.94072927	0.94056695	0.94001504	0.93970121	0.94169624	0.94054174
8	0.94045873	0.9401936	0.93972285	0.93938738	0.94137158	0.94022682
9	0.9410431	0.94085372	0.94026394	0.94000422	0.94203173	0.94083934
10	0.94100523	0.94076174	0.94017736	0.93995011	0.94195056	0.940769

Table 7.42: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
90 days 6 hours	11 hr 56 min 56 sec

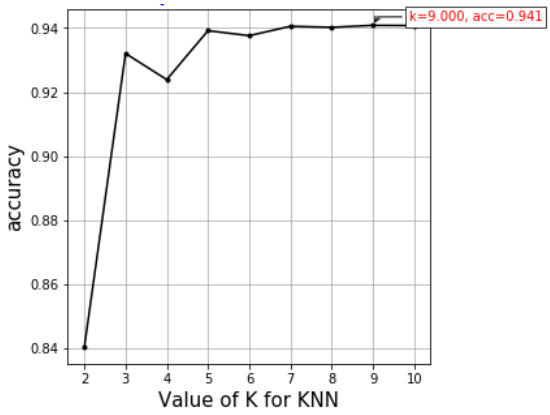

Figure 7-22: K vs accuracy graph for Kulczynski distance similarity measure

4. Result for Chebyshev Distance

Table 7.43: Result for Chebyshev Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.84017899	0.83838258	0.84084994	0.84086076	0.84207393	0.8404692
3	0.93207187	0.93172017	0.93173099	0.93141175	0.93334163	0.93205528
4	0.92364173	0.92318722	0.9236742	0.92357139	0.92543071	0.92390105
5	0.93938196	0.93899779	0.93879218	0.93834849	0.94038136	0.93918035
6	0.93803466	0.93727172	0.93725549	0.93689837	0.93878511	0.93764907
7	0.94075633	0.94055071	0.94013949	0.93982025	0.94177741	0.94060883
8	0.94041003	0.94027476	0.9397986	0.93948477	0.9414798	0.94028959
9	0.9410431	0.94083208	0.94033428	0.94002045	0.9420696	0.9408599
10	0.94102146	0.94077256	0.94026935	0.93995552	0.94198844	0.94080146

Table 7.44: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
53days 1hr 15mins	9hrs 50mins 30sec

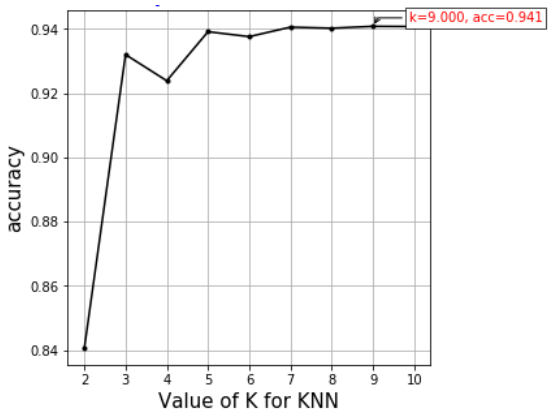

Figure 7-23: K vs accuracy graph for Chebyshev distance similarity measure

5. Result for Cosine Similarity

Table 7.45: Result for Cosine Similarity

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.93873807	0.90624577	0.90734959	0.92381488	0.92863945	0.92095755
3	0.9410918	0.94077797	0.94037757	0.94007456	0.92875308	0.93821499
4	0.94096735	0.94069681	0.91137528	0.92521088	0.92872061	0.92939418
5	0.9410918	0.94094571	0.94037757	0.94010161	0.94212371	0.94092808
6	0.9410918	0.94094571	0.94037757	0.94007456	0.94136617	0.94077116
7	0.9410918	0.94094571	0.94037757	0.94010161	0.94212371	0.94092808
8	0.9410918	0.94094571	0.94037757	0.94010161	0.94211289	0.94092591
9	0.9410918	0.94094571	0.94037757	0.94010161	0.94213453	0.94093024
10	0.9410918	0.94094571	0.94037757	0.94010161	0.94212912	0.94092916

Table 7.46: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
97days 18hrs 5mins	9hrs 53mins 48sec

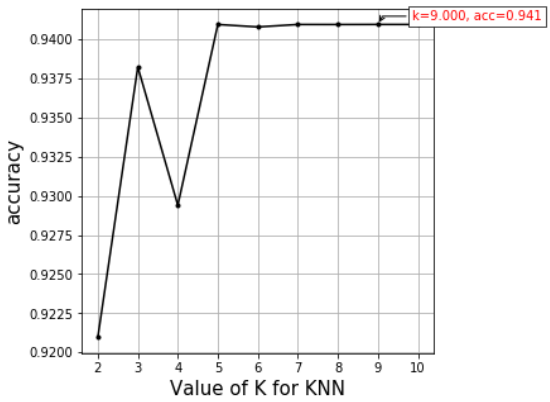

Figure 7-24: K vs accuracy graph for Cosine distance similarity measure

6. Results for Soergel Distance

Table 7.47: Results for Soergel Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.84030885	0.83862065	0.84106637	0.84003289	0.8415653	0.84031881
3	0.93215845	0.93180133	0.93192037	0.93113579	0.93329834	0.93206285
4	0.92381488	0.9234253	0.92379865	0.92335495	0.92513311	0.92390537
5	0.93929539	0.93920341	0.93872725	0.93837554	0.94026773	0.93917386
6	0.93793185	0.9374611	0.93702282	0.93690378	0.938731	0.93761011
7	0.94072927	0.94056695	0.94001504	0.93970121	0.94169624	0.94054174
8	0.94045873	0.9401936	0.93972285	0.93938738	0.94137158	0.94022682
9	0.9410431	0.94085372	0.94026394	0.94000422	0.94203173	0.94083934
10	0.94100523	0.94076174	0.94017736	0.93995011	0.94195056	0.940769

Table 7.48: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
7985440.3 seconds	42644.66 seconds

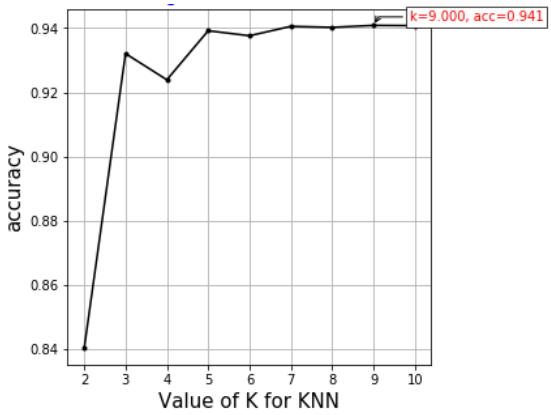

Figure 7-25: K vs accuracy graph for Sorgel distance similarity measure

7. Results for Sorenson Distance

Table 7.49: Results for Sorenson Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.84030885	0.83862065	0.84106637	0.84003289	0.8415653	0.84031881
3	0.93215845	0.93180133	0.93192037	0.93113579	0.93329834	0.93206285
4	0.92381488	0.9234253	0.92379865	0.92335495	0.92513311	0.92390537
5	0.93929539	0.93920341	0.93872725	0.93837554	0.94026773	0.93917386
6	0.93793185	0.9374611	0.93702282	0.93690378	0.938731	0.93761011
7	0.94072927	0.94056695	0.94001504	0.93970121	0.94169624	0.940541742
8	0.94045873	0.9401936	0.93972285	0.93938738	0.94137158	0.940226828
9	0.9410431	0.94085372	0.94026394	0.94000422	0.94203173	0.940839342
10	0.94100523	0.94076174	0.94017736	0.93995011	0.94195056	0.940769

Table 7.50: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
7078337.9 seconds	41061.559248209 seconds

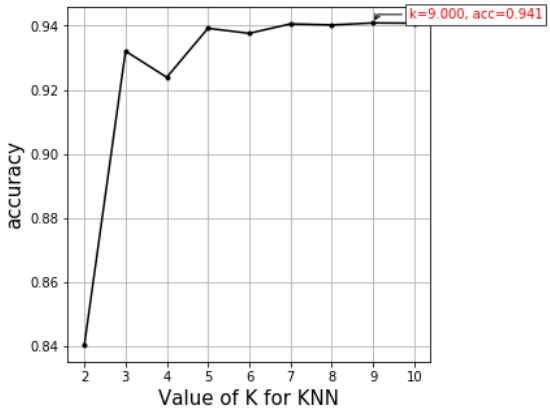

Figure 7-26: K vs accuracy graph for Sorenson distance similarity measure

8. Results for Tanimoto Distance

Table 7.51: Results for Tanimoto Distance

K	Accuracy (1st Fold)	Accuracy (2nd Fold)	Accuracy (3rd Fold)	Accuracy (4th Fold)	Accuracy (5th Fold)	Average Accuracy
2	0.84030885	0.83862065	0.84106637	0.84003289	0.8415653	0.84031881
3	0.93215845	0.93180133	0.93192037	0.93113579	0.93329834	0.93206285
4	0.92381488	0.9234253	0.92379865	0.92335495	0.92513311	0.92390537
5	0.93929539	0.93920341	0.93872725	0.93837554	0.94026773	0.93917386
6	0.93793185	0.9374611	0.93702282	0.93690378	0.938731	0.93761011
7	0.94072927	0.94056695	0.94001504	0.93970121	0.94169624	0.94054174
8	0.94045873	0.9401936	0.93972285	0.93938738	0.94137158	0.94022682
9	0.9410431	0.94085372	0.94026394	0.94000422	0.94203173	0.94083934
10	0.94100523	0.94076174	0.94017736	0.93995011	0.94195056	0.940769

Table 7.52: Time Comparison betwenn CPU and GPU

CPU Time	GPU Time
6098829 Seconds	40016.88 Seconds

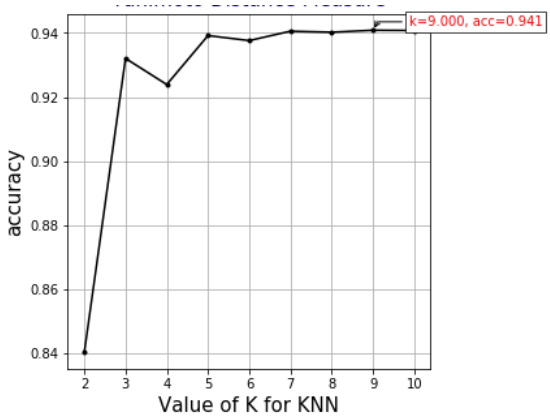

Figure 7-27: K vs accuracy graph for Tanimoto distance similarity measure

7.5.2.4 Accuracy Comparison

The graph plots for the accuracy comparison is shown below.

1. Accuracy of Binary and Multi-Class Classification on Ransomware Dataset

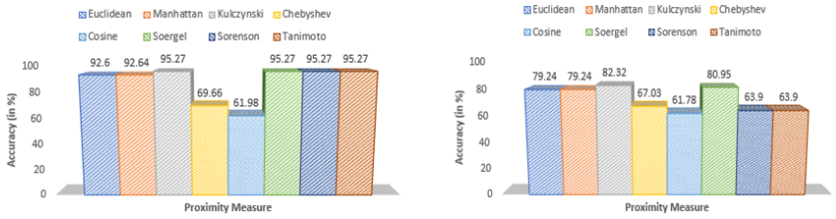

Figure 7-28: Accuracy of Binary and Multi-Class Classification on Ransomware Dataset

2. Accuracy of SWaT Dataset

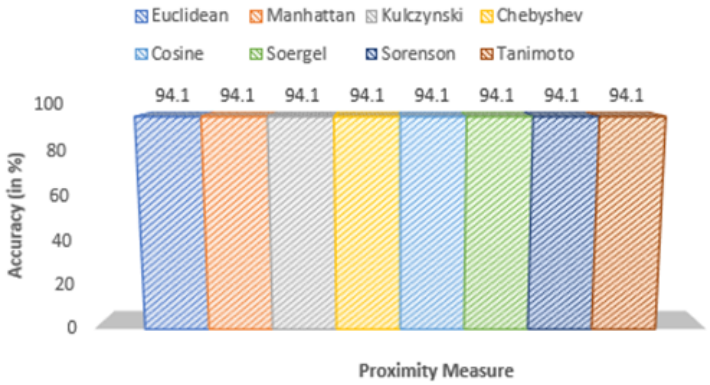

Figure 7-29: Accuracy SWaT Dataset

7.5.2.5 Time Comparison

1. CPU vs GPU Time Comparison for Binary Ransomware Dataset

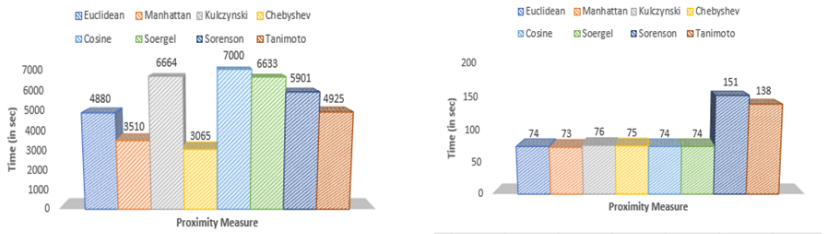

Figure 7-30: CPU vs GPU Time Comparison for Binary Ransomware Dataset

2. CPU vs GPU Time Comparison for Multi-Class Ransomware Dataset

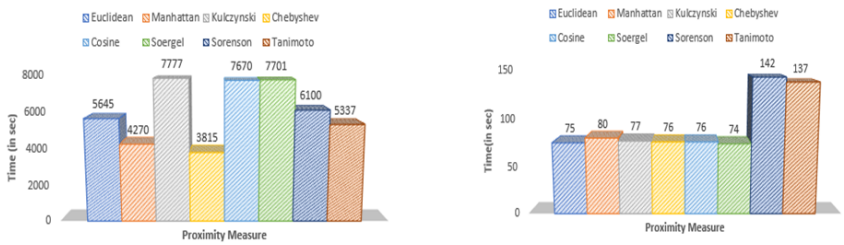

Figure 7-31: CPU vs GPU Time Comparison for Multi-Class Ransomware Dataset

3. CPU vs GPU Time Comparison for SWaT Dataset

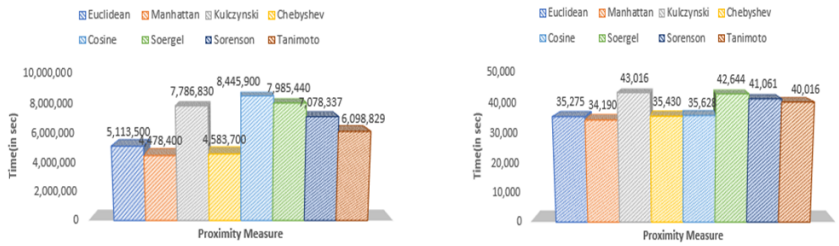

Figure 7-32: CPU vs GPU Time Comparison for Swat Dataset

Table 7.53: A ratio of CPU time and GPU time

S. No	Proximity measures	Binary classification of Ransomware Dataset	Multi class classification of Ransomware Dataset	Classification of SWaT Dataset
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

7.6 Discussion

In this chapter, the TUKNN algorithm, a parallel version of the k-Nearest Neighbors (k-NN) model, is implemented. It has been found that the Kulczynski distance and Soergel distance, when implemented with the k-NN model, give an accuracy of 95% for binary classification of the ransomware dataset. In contrast, Euclidean and Manhattan distances give an accuracy of 93%. For multi-class