I.J. Intelligent Systems and Applications, **2021, 1, 34-44**
Published Online February 2021 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2021.01.03

An Enhanced Adaptive *k*-Nearest Neighbor Classifier Using Simulated Annealing

Anozie Onyezewe
Department of Computer Science, Ahmadu Bello University, Zaria, Nigeria
E-mail: aonyezewe@gmail.com

Armand F. Kana, Fatimah B. Abdullahi and Aminu O. Abdulsalami
Department of Computer Science, Ahmadu Bello University, Zaria, Nigeria
E-mail: {donfackkana, zeeh429, lecturer34}@gmail.com

Received: 02 November 2019; Accepted: 13 January 2020; Published: 08 February 2021

Abstract: The k-Nearest Neighbor classifier is a non-complex and widely applied data classification algorithm which does well in real-world applications. The overall classification accuracy of the k-Nearest Neighbor algorithm largely depends on the choice of the number of nearest neighbors(*k*). The use of a constant *k* value does not always yield the best solutions especially for real-world datasets with an irregular class and density distribution of data points as it totally ignores the class and density distribution of a test point's k-environment or neighborhood. A resolution to this problem is to dynamically choose *k* for each test instance to be classified. However, given a large dataset, it becomes very tasking to maximize the k-Nearest Neighbor performance by tuning *k*. This work proposes the use of Simulated Annealing, a metaheuristic search algorithm, to select optimal *k*, thus eliminating the prospect of an exhaustive search for optimal *k*. The results obtained in four different classification tasks demonstrate a significant improvement in the computational efficiency against the *k*-Nearest Neighbor methods that perform exhaustive search for *k*, as accurate nearest neighbors are returned faster for *k*-Nearest Neighbor classification, thus reducing the computation time.

Index Terms: Adaptive Algorithms, Classification, Heuristic Learning, k-Nearest Neighbor (kNN), Parameter Optimization.

1. Introduction

The *k*NN algorithm is one of the simplest classification algorithms and one of the most used learning algorithms due to its simplicity, ease of implementation and non-parametric principle. Before the rule was introduced for classification in Cover and Hart [1], the NN rule was mentioned in Nilsson [2] as "minimum distance classifier" and Sebestyen [3] as "proximity algorithm". The *k*NN method is an instance-based learning algorithm [4] which performs classification on a data point by determining parameter *k* (number of nearest neighbors), calculating the distance between the test instance and all the training examples and sorting the distance to determine nearest neighbors based on the *k*th minimum distance, gathering the class of the nearest neighbors and using the majority of the class of nearest neighbors as the prediction value of the test instance. The *k*NN algorithm are termed instance-based, or lazy learners because they make decisions by comparing the training set with the test set for each classification they perform. They simply store the instances and they do not do any work on instances until it is given a test tuple. Other methods for classification such as rule-based classification, Decision Tree induction, Classification by Back-propagation, Bayesian classification, and Support Vector Machines (SVM) are examples of non-instance-based learners, otherwise known as eager learners [4,5,6].

The success of the *k*NN algorithm greatly depends on the choice of parameter *k*. If *k* is too large, large classes will overpower smaller classes. On the flip side, if *k* is too small, the result will be influenced by only a small number of neighbors. To proffer a solution to this problem, Li [10] proposed the use of different *k* values for different classes against the use of constant *k* value for all classes.

Most practical datasets have uneven density and class distribution. The density distribution can be considered as dense if the distance between data points are very close, and sparse if data points are farther apart. Euclidean distance function is popularly used by most *k*NN algorithms to find the distance between data points. Due to the variations in density, using different *k* values for different points will be very useful as a simple intuition will be to consider more neighbors in a dense neighborhood and fewer neighbors in a sparse neighborhood as using a constant *k* value will lead to bias on larger classes. Thus, for *k*NN variants which finds optimal *k* ($1 \leq k \leq \text{size of dataset}$) dynamically, finding optimal *k* can become a very challenging, computationally expensive task.

Many techniques have been put forward in order to enhance the *k*NN computation time. One of such method is reducing the number of training points without trading-off precision. Zhou, using this approach, proposed a fast KNN text classification approach based on trimming the training data [7]. Another method is to adopt the fast algorithm using a fitness function as proof of classification correctness. A traditional way to achieve this is by indexing the training set using *kd*tree [8] which stores a set of points in k-dimensional space, where k is the number of attributes. In this paper, we propose the EACKER method which adopts the Simulated Annealing (SA), a metaheuristic search algorithm, to return nearest points to a test instance faster than *k*NN methods that perform an exhaustive search for optimal *k*, without trading off accuracy.

2. Related Works

In recent years, the traditional *k*NN algorithm has been enhanced for better performance using hyperparameter tuning techniques, hybridization, or both.

Berchtold *et al*. [9] Proposed a Voronoi cells-based method which gives high efficiency for uniformly distributed as well as for real datasets to overcome the problem of traditional *k*NN's memory limitation. Traditional *k*NN use too much memory in searching '*k*' nearest neighbors of a test instance when high dimensional datasets are fed to it.

Li *et al*. [10] proposed an enhanced *k*NN method, where different values of *k* are used for different classes, against the use of a constant *k* for all classes. Additional instances were employed to decide if a test instance should be classified to classes that have more instances in the training set

Jiang *et al*. [11] proposed a dynamic *k*NN Naïve Bayes with Attribute weighted algorithm. In their work, lazy learning and eager learning techniques were combined to improve the efficiency of the *k*NN classifier. Their method learns the optimal '*k*' value eagerly during the training period of the model.

Yang *et al.* [12] proposed the use of metaheuristic method for finding closest points to a test point quickly. Inverted array was used to index instances while Simulated Annealing algorithm was employed to search the expected neighbors faster. Their method demonstrates the introduction of metaheuristic search algorithms in reducing the computation time of *k*NN classification and their experimental results show a substantial enhancement in computational efficiency compared to the conventional *k*-NN.

Sun and Huang [13] brought forward an adaptive *k*-nearest neighbor algorithm (AdaNN) that resolves the limitation of the *k*NN. Best value of *k* is searched from 1-NN to 9-NN. If the correct class label of a training sample cannot be found by using 1-NN to 9-NN, the algorithm makes 9 its optimal value for *k*.

Liu & Chawla [14] Defined a class confidence weighting strategy to handle imbalanced datasets. It uses the probability of attribute values of given class labels to weight samples in *k*NN. It reduces the biasing towards majority class prediction in the traditional *k*NN.

Kuhkan [15] proposed an algorithm that is centered on dynamic weighting to improve the *k*NN algorithm's classification. Based on the importance of features, certain weights are assigned to these features so as to avoid similar effect of all features during classification. The results show substantial enhancement of the classification accuracy when compared with some other classification algorithms on 10 different datasets.

Mullick *et al.* [16] Proposed AdaKNN to find optimal k and handle classs imbalance. For each training point, the algorithm finds a single correctly classifying k value, and feeds them to a Multi Layer Perceptron to generate it's k-terrain. A Global Imbalance Handling Scheme (GIHS) is used in classification to handle class imbalance. The results show competitive performance of AdaKNN against the *k*NN and some other newer variants of *k*NN (NWkNN, kPNN, kENN, WkNN, and *dyn*KNN).

Kibanov *et al*. [17] proposed an adaptive kNN classifier based on expected accuracy where optimal *k* is dynamically selected for each instance to be classified, such that the highest probability of classification accuracy is attained. They introduce expected accuracy measure which is defined as "the accuracy of a set of structurally similar observations" to improve the accuracy of *k*NN classification. Similarity functions were employed to discover these "observations" for which they used five (5) different classification tasks based on geo-spatial data. Their sequential iteration over a large range of *k* values (*k* = 1 to 200) for finding optimal *k* increases runtime and space, thus efficiency is reduced. Also, the classifier was applied only on geo-spatial data using the *lat-lon* similarity function.

3. Adaptive *k*-Nearest Neighbor

Adaptive *k*-NN is a modification over the traditional *k*NN method for achieving higher classification accuracy. *k*NN is a simple and basic classification algorithm with some advantages: *k*NN is a lazy learner, thus has no training time. Secondly, it can be used as incremental learner. On the other side, *k*NN has some disadvantages: noise tolerance which could lead to low classification accuracy. The concept of the adaptive *k*NN algorithm is to dynamically select optimal *k* for every test instance (as against the use of a constant *k* in the traditional *k*NN) and to use this *k* with standard *k*NN in order to maximize classification accuracy.

For this paper, the ACKER algorithm developed in [17] is adopted as the research basis for adaptive *k*NN algorithms. We introduce the use of the *kd*-Tree structure for indexing data points while we employ the SA algorithm to

return the nearest neighbors to a query point for faster classifications. Given a query (test) point, the ACKER algorithm estimates the expected accuracy for different possible values of *k* and chooses a *k* that provides a maximum value of the expected accuracy.

Algorithm I: ACKER [17]

Data: query instance *p*, function *f*, set *Range*, integer *l*
Result: Class of instance *p*

1. $optimal\_k$ = 0
2. $max\_accuracy$ = -1
3. **for** *k* in *Range* **do**
4. $candidate\_accuracy = expected\_accuracy(p, k, f, l)$
5. **if** $candidate\_accuracy > max\_accuracy$ **then**
6. $optimal\_k$ = k
7. $max\_accuracy = candidate\_accuracy$
8. **end**
9. **end**
10. $predicted\_class = kNN(p, optimal\_k)$
11. **return** ($predicted\_class$)

3.1. Similarity Functions

The similarity function $sim: P \times P \times N \rightarrow R$ defines the similarity of the k-environment of two given points. Simply stated as $sim: P \times P \rightarrow R$. Considering a function *f*: P ×N → R that characterizes a point and its k-environment. Then, a similarity function based on *f* as defined in [17], is shown as:

$$sim_f: P \times P \times N \rightarrow R \quad (p, p', k) \rightarrow -dist(f(p,k), f(p',k)) \tag{1}$$

Where $dist(x, y)$ indicates the Euclidean distance between two points. The standard Euclidean distance is used to compute the closest neighbors of an instance. Let the feature vector $(a_1(x), a_2(x), \ldots a_n(x))$ represent an arbitrary instance *x*. Where $a_i(x)$ indicates the value of the *i*th attribute of instance *x*. Then the Euclidean distance between two instances *x* and *y* is defined to be $d(x, y)$, where

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2}$$

The higher the similarity of two points are with their k-environments, the higher the value of sim_f. $P_{sim,l}(p, k)$ is defined as "the set containing the *l* points whose k-environments are most similar to the *k*-environments of *p* with respect to the given similarity function", also written as

$$P_{sim,l}(p, k) = argmax_p^l, (sim(p, p', k)) \tag{3}$$

The similarity functions used in [17] and adopted in this paper are introduced below:

A. Average Distance Similarity Function (Sim_{avg_dist})

The average distance similarity function $avg\_dist(p, k)$ which is "the average distance to the k nearest neighbors of query point *p*" is denoted as:

$$avg\_dist(p, k) = \frac{\sum_{i=1}^{k} dist(p, p_i)}{k} \tag{4}$$

Where p_i is defined here and after, as "the *i*th nearest neighbor to p". The similarity function based on avg_dist is defined as:

$$sim_{avg_{dist}}(p, p', k) = -dist(avg\_dist(p, k), avg\_dist(p', k)) \tag{5}$$

B. Maximum Distance Similarity Function (Sim_{max_dist})

The maximum distance similarity function $max\_dist(p, k)$ which is "the maximum distance to the *k*th nearest neighbor of the query point *p*" is denoted as:

$$max\_dist(p,k) = dist(p,p_k) \tag{6}$$

The similarity function based on max_dist for two points p and p' is defined as follows:

$$sim_{\max_{dist}}\ (p,p',k) = -dist(max\_dist(p,k), max\_dist(p',k)) \tag{7}$$

C. Maximum_Average_Combined Distance Similarity Function ($Sim_{max\_avg\_comb}$)

The maximum-average-combined distance similarity function $max\_avg\_comb(p,k)$ which returns a tuple of results of avg_dist (4) and max_dist (6) is denoted as:

$$max\_avg\_comb(p,k) = (max\_dist(p,k),\ avg\_dist(p,k)) \tag{8}$$

The similarity function based on max_avg_comb for two points p and p' is defined as follows:

$$sim_{max\_avg\_comb}(p,p',k) = -dist(max\_avg\_comb(p,k), max\_avg\_comb(p',k) \tag{9}$$

D. Latitude_Longitude Distance Similarity Function (Sim_{lat_lon})

The latitude-longitude distance similarity function is denoted as:

$$lat_{lon(p)} = \big(lat(p), lon(p)\big) \tag{10}$$

The similarity function based on lat_lon is defined as the Euclidean distance between an instance (point) and its neighbors:

$$sim_{lat\_lon}(p,p') = -dist(lat\_lon(p), lat\_lon(p')) \tag{11}$$

3.2. Expected Accuracy

The concept of expected accuracy is: Given a query instance *p* along with its "k-environment", a set of points $P_{sim,l}(p,k)$ with similar k-environments from the training set is found. As the "k-environments" are similar, it is assumed that there is a correlation between the accuracy of the *k*NN-based classification function for given point *p*, denoted as $acc(c'_k(p),\{p\})$, and the accuracy of *k*NN-based classification function for $P_{sim,l}(p,k)$ [12]. Thus, the expected accuracy is defined as:

$$exp\_acc_l(p,k) = acc(c'_k, P_{sim,l}(p,k) \tag{12}$$

It is supposed that different regions are statistically similar. Thus, it is established that expected accuracy is a good indicator for a real accuracy.

Algorithm II: Expected Accuracy [17]

Data: query instance *p*, function *f*, integer *k*, integer *l*
Result: "Expected accuracy for $KNN(p,k)$"

1. find set $P_{sim,l}(p,k)$ of *l* different points *p'* with minimal difference to reference function w.r.t *f*:
   $sim(p,p') = |f(p,k) - f(p',k)|$,
   $P_{sim,l}(p,k) = argmax(\Sigma sim(p,p',k))$ as in (3)
2. $correctly\_predicted$ = number of correct predictions of $KNN(p',k)$,
   where $p' \in P_{sim,l}(p,k)$
3. $expected\_accuracy = \frac{correctly\ predicted}{l}$
4. **return** $expected\_accuracy$

4. *k*-Nearest Neighbor with Simulated Annealing

The *k*-nearest neighbor algorithm assumes all instances to correspond to points in an n-dimensional space [12]. The closest neighbors to a point are defined in terms of the standard Euclidean distance as shown in (2). The target of *k*NN is to get the *k* closest values of all distances between the query point and the other training points, defined as:

$$min_k d(x,y) = -\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad (13)$$

From (13), it is deduced that this becomes a problem of combinatorial optimization and $\sum_{i=1}^{n} x_i - y_i$ has to be computed instantaneously. Simulated Annealing (SA) method is a good solution in the field of combinatorial optimization based on correlation with the physical process of annealing in metallurgy [12], therefore the idea of Simulated Annealing is adopted to improve the computational efficiency of the adaptive *k*NN methods. Since data storage structure is very related with algorithms, the *kd*-tree, which will be used in this paper as the storage structure, is introduced.

4.1. Data Storage

The *kd*-tree which is used as our data storage for indexing points is a b+ tree in which every leaf node is a k-dimensional point, with every non-leaf node generating a splitting hyperplane that divides the space into two half-spaces. Fig.1 shows how points (P1 … P*n*) represented in an array (b) are indexed using the *kd*-tree (a). The points to the left of the hyperplane are represented by the left subtree of that node (c), (e), (g) and the points to the right of the hyperplane are represented by the right subtree (d), (f), (h), with each point in the nodes referencing the instances in an array. The hyperplane direction of points in a *kd*-tree is chosen in the following steps:

a) Pick a random dimension

b) Find the median of points in the chosen axis and split them into two half-spaces. Points ≥ median are represented in the right subtree while points < median are represented in the left subtree

c) Repeat until the *k*th dimension is split.

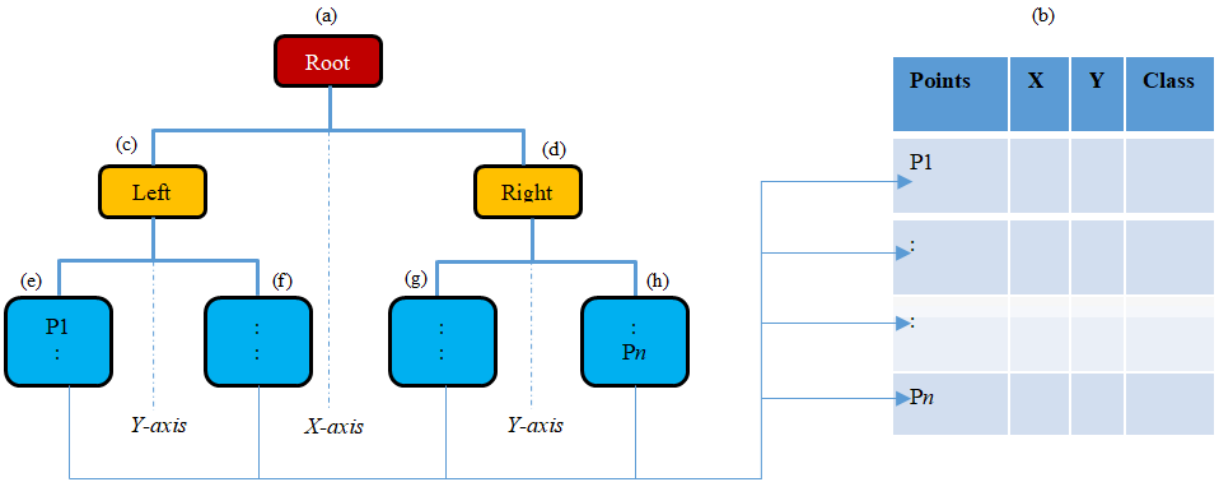

Fig.1. Storage structure

4.2. Simulated Annealing

This research work seeks to minimize the running time of the ACKER method whilst maintaining its accuracy by using a metaheuristic method; the Simulated Annealing (SA) algorithm, to find the accurate nearest neighbors to a given point in fewer iterations as against the exhaustive, brute-force approach. The concept of simulated annealing is a random-search metaheuristic technique for finding a good solution to an optimization problem in a large search space by trying random variations of the current solution [12]. The choice of SA to find nearest neighbors comes from its advantages, as it is a very good metaheuristic for combinatorial optimization (CO) problems, and its capability to escape from being stuck in local minima.

Algorithm III: *SA Pseudocode [18]*

Input: Initial Temperature, Final Temperature, Cooling rate
Result: Best solution

1 Generate an initial solution X_0
2 **Do**
3 Generate new solution X_0 in the neighborhood of X_0
4 Compute the acceptance probability
5 Decide on acceptance or rejection of new solution
6 Memorize the best solution found so far
7 Reduce the temperature
8 **While** stopping condition is not exceeded

4.3. Annealing Schedule

The traditional implementation of the Simulated Annealing algorithm is one in which homogeneous Markov chains of finite length are generated at decreasing temperatures [12]. The process starts with an initial solution X, which in this case is finding the k points in the same leaf node as the query point, and creates and updated solution X' in the neighborhood of the current solution which is finding the next k points in the same node or neighboring leaf nodes. Solutions are generated if the fitness value $F(x')$ is less than $F(x)$ [18].

We define our fitness functions $F(x)$ to be: computing the Euclidean distance of the query point and the first k points in that leaf node, and $F(x')$ to be: computing the Euclidean distance of the query point and the next k points in that leaf node and the neighboring leaf nodes. However, a greater fitness of x' is sometimes accepted with a probability so as to enable the algorithm avoid being trapped in local minima. The probability used in this work is adopted from [18] and defined in (14)

$$P_r = \exp\left(\frac{-(f(x')-f(x))}{T}\right) \tag{14}$$

Where T is the temperature or the control parameter. Our cooling process is finding the next k points and computing the Euclidean distance with the query point. If these new distances don't substitute the already existing ones, then it is concluded as the final temperature. We set the length of our Markov chain as k.

5. EACKER

The EACKER method proposed in this paper is a modification of the ACKER method in [17] which finds optimal k for a query point based on the kNN accuracy of similar points, where k is dynamically selected for each point to be classified. The EACKER aims at improving the computational efficiency of the ACKER by replacing the brute-force approach for finding optimal k with the use of metaheuristic search method, thus reducing the time taken for classification. Datasets are preprocessed and split into training(X) and test(Y) sets with 70% and 30% for train and test set respectively. The training datasets are indexed and stored in a k-dimensional tree (*kd*-tree).

Algorithm IV: EACKER

Data: query point *p*, set *Range*, function *f*, integer *l*
Result: Class of point *p*

1. $optimal\_k$ = 0
2. $mode$ = highest occurring number of k' from set of similar points $P_{sim,l}(p,k)$
3. find set k' of correctly classifying k values $KNN(p,k)$ for all training points in $P_{sim,l}(p,k)$ using Simulated Annealing
4. **for** k in *Range* **do**
5. find set $P_{sim,l}(p,k)$ of l different points p' with minimal difference to reference function w.r.t f: $sim(p,p') = |f(p,k) - f(p',k)|$, $P_{sim,l}(p,k) = argmax(\Sigma sim(p,p',k))$ see (3)
6. **if** $mode > 1$ **then**
7. $optimal\_k$ = random $mode$
8. **end**
9. $optimal\_k = mode$
10. **end**
11. $predicted\_class = kNN(p, optimal\_k)$

return $(predicted\_class)$

At the classification phase, a set of similar points($P_{sim,l}$) is computed for all query points from test dataset using a similarity function (as shown in section III), where $l = \sqrt[3]{n}$, (where n = total instances in the dataset) and a set of correctly classifying k values(k') computed for all instances (points) with the aid of simulated annealing, for k = $1 \ldots k_{max}$. Where $k_{max} = \sqrt{n}$. which satisfies the properties: a) $k$max "should neither be too large nor too small to preserve the local structure of the neighborhood." b) $k$max "should be data-dependent" [16]. The mode from the set of correctly classifying k values (k') for each training instance in $P_{sim,l}$ is then returned and used as optimal k for classifying the query point in the test dataset. It is expected that the most occurring k' value will yield the highest expected accuracy.

The computation for expected accuracy by finding similar points for each test instance has been proven to yield more accurate results over the traditional kNN [17], and thus we adopt this method in our algorithm. However, the

method of finding the similar points for each point to be classified greatly affects the computation time for finding the correctly classifying k values k' and optimal k for each test point. The prospect of an exhaustive search for similar points and k' is eliminated by introducing the SA metaheuristic search which returns k' faster, while the mode function returns the optimal k from the set of k' returned. This significantly reduces the computation time without accuracy trade-off.

6. Evaluation

The details of the system implementation, evaluation and analysis of results is discussed in this section. The environment for which experiments were carried out is: Windows 10 Home 64-bit Operating System, x64-based processor with Intel(R) Core i5(8^{th} Gen). CPU speed at 1.80 GHz 3.1 GHz turbo boost 8.00GB RAM and 1TB HDD. The algorithm is developed using Anaconda navigator and Jupyter notebook version 5.0.0.

6.1. Data Source

For this research, four different datasets that represent geo-spatial data and non-geo-spatial data from three different open source benchmark dataset repositories were used. All experimental results presented were conducted using 10-fold cross-validation where we used for each experiment, 70% of the data as training data and 30% of the data as test data. From these datasets we constructed 4 different classification tasks to evaluate the EACKER approach. The datasets and their classification tasks are presented below:

Milan Twitter Dataset. The Milan Twitter dataset (*Social-Pulse Milano Spazio-Dati*) was gathered in Milan, Italy from November to December, of 2013. For the classification tasks, tweets where one of the seven most popular local geographical items in Italy is mentioned are considered. Also, in order to add some noise to the data, a general entity ("Italy") is added [17]. The dataset has a total of 269,290 records (*https://dandelion.eu/products/- datatxt/*).

San Francisco Crimes Dataset. This dataset contains all coordinates and categories of crimes committed in San Francisco in 2015. Given the coordinates of a particular crime, the job of the classification model is predicting the crime category [17]. The dataset contains over 2.2m records (*https://data.sfgov.org/Public-Safety/SFPD-Incidents-Previous-Year-2015-/ritf-b9ki)*.

Iris Dataset. The Iris flower dataset is a multivariate dataset introduced by the British statistician and biologist Ronald Fisher. The data set consists of 50 samples from each of three species of the Iris flower (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals, and the length and width of the petals, in centimeters. The task of the classifier is to predict the specie of an Iris flower, given the length and width of the sepals and petals of that Iris flower.

Avila Dataset. This dataset contains an extraction of 800 images from the Avila Bible. For the prediction task each pattern is assigned to one of the 12 classes: A, B, C, D, E, F, G, H, I, W, X, Y. the dataset has been normalized using the Z-normalization method and divided into two sets: a training set which contains 10430 samples, and a testing set which contains 10437 samples.

6.2. Result for Data Storage in kd-Tree

As stated earlier, the *kd*-tree is a b+ in which every leaf node is a k-dimensional point. The runtime complexity of Search, Insert and Delete operations in a *kd*-tree are: $O(log\ n)$ average case and $O(n)$ worst case.

6.3. Nearest Neighbor Search in kd-Tree using Simulated Annealing

Here, we show the runtime complexity of finding *k* nearest neighbors and computing similar points ($P_{sim,l}$) for any given point and its k-environment. Based on that, we estimate the algorithmic runtime of the EACKER algorithm.

6.4. Complexity of finding Nearest Neighbors

The complexity of finding the nearest neighbors of any given point is the same as the complexity of searching in *kd*-trees. The introduction of Simulated Annealing aids in finding accurate nearest points to any given point by comparing the Euclidean distance of that point to other neighboring points. Thus, regardless of the number of points to be searched or compared with, the complexity remains $O(log\ n)$.

6.5. Complexity of finding Similar points ($P_{sim,l}$)

The elements in *kd*-trees are not linked, therefore it is necessary to search for l points in ($P_{sim,l}$). In the average case, this will be performed in $O(l \times log\ n)$, or $O(n.log\ n)$.

6.6. Complexity of EACKER

Based on the running time complexities of finding the nearest neighbors and similar points of any given point, we estimate the algorithmic runtime complexity to be $O(n.(log\ n)^2)$.

6.7. Experimental Results

In this section, we discuss the performance of the traditional *k*NN, ACKER and EACKER, and show the experimental results obtained from four different datasets earlier introduced with different *k* values using the traditional *k*NN, ACKER and EACKER methods in terms of the CPU running time (in seconds) and classification accuracy. Each value reported is the average value obtained from running each experiment ten times over using the average distance similarity function (sim_{avg_dist}). We also show our choice range of *k* values and how they affect the computation time and the classification accuracy on the classification tasks.

Table 1 shows the CPU running time using different *k* values from 5 to 1,000 on four different datasets using the *k*NN, ACKER and EACKER methods.

It can be observed that the *k*NN method runs on a constant time, the ACKER method exhibits polynomial runtime, and the EACKER method demonstrates a polylogarithmic runtime which is logarithmic to ACKER.

It can also be observed that there are no figures for the CPU running time for the *Iris* classification task after *k=100* for *K*NN and *k=50* for ACKER and EACKER respectively. This is as a result of the number of instances (150) contained in the *Iris dataset*, therefore, a greater number of *k* cannot be considered after 150 for *k*NN, and 50 for ACKER and EACKER where *k* and similar points, *l* are to be found.

Table 2. shows the classification accuracies using different *k* values from 5 to 1,000 on four different datasets using the *k*NN, ACKER and EACKER methods. The ACKER and EACKER's classification performance in terms of accuracy clearly outperforms the traditional *k*NN method. However, ACKER's classification performance in terms of accuracy does not differ from EACKER's because both methods make use of the same similarity functions to return the same optimal *k*.

6.8. EAKER Accuracy Dependent on Similarity Functions

Fig. 2-5 show the classification accuracies using different *k* values from 5 to 1,000 on four different datasets using the ACKER and EACKER methods on different similarity functions. The figures show how different similarity functions influences the overall classification accuracy for each dataset.

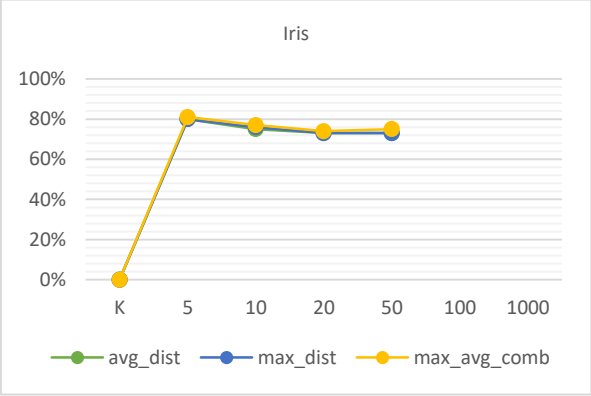

Fig.2. Similarity function accuracy on Iris dataset

It can be observed that there isn't a substantial variance in the Iris and Avila datasets, although the max_avg_comb similarity function marginally outperforms the rest. The Milan-Twitter dataset also show competitive accuracy results for all similarity functions, while the lat_lon similarity function clearly outperforms the other similarity functions in the San Francisco Crimes dataset.

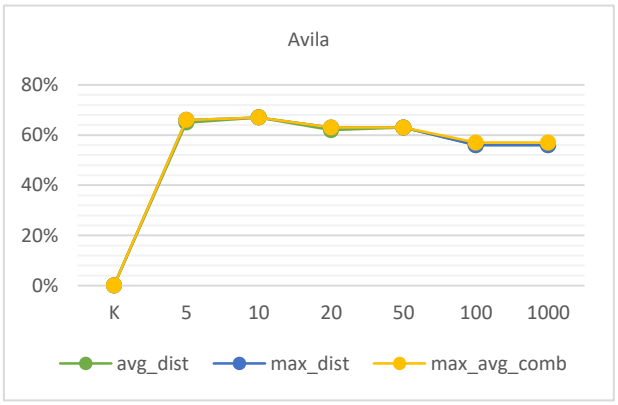

Fig.3. Similarity function accuracy on Avila dataset

Fig.4. Similarity function accuracy on SF-Crimes dataset

Table 1. Comparative Analysis for Running Time (Secs)

	KNN				ACKER				EACKER			
K	*Milan-Twitter*	*SF-crimes*	*Iris*	*Avila*	*Milan-Twitter*	*SF-crimes*	*Iris*	*Avila*	*Milan-Twitter*	*SF-crimes*	*Iris*	*Avila*
5	23	20	0.2	20	24	20	0.6	32	10.4	10.3	0.1	15
10	23	20	0.2	20	44	40	0.8	62	25	20	0.3	30
20	23	20	0.2	20	84	79	1.1	122	52	42	0.6	75
50	23	20	0.2	20	214	198	2.0	310	166	126	1.0	150
100	23	20	0.2	20	435	398	-	652	339	253	-	466
1,000	23	20	-	20	4106	3924	-	5402	3090	2230	-	4360

Table 2. Comparative Analysis for Accuracy

	KNN				ACKER				EACKER			
K	*Milan-Twitter*	*SF-crimes*	*Iris*	*Avila*	*Milan-Twitter*	*SF-crimes*	*Iris*	*Avila*	*Milan-Twitter*	*SF-crimes*	*Iris*	*Avila*
5	88%	15%	90%	59%	90%	20%	80%	65%	90%	20%	80%	65%
10	88%	20%	83%	58%	88%	25%	75%	67%	88%	25%	75%	67%
20	88%	26%	79%	55%	87%	31%	73%	61%	87%	31%	73%	62%
50	88%	28%	76%	51%	87%	33%	73%	63%	87%	33%	73%	63%
100	88%	28%	75%	48%	86%	33%	-	56%	86%	33%	-	56%
1,000	88%	28%	-	40%	86%	33%	-	56%	86%	33%	-	56%

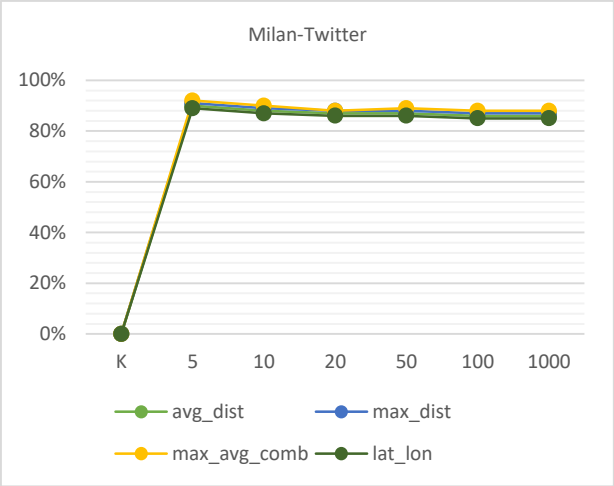

Fig.5. Similarity function accuracy on Milan-Twitter dataset

7. Conclusion and Further Work

This work successfully applied its classification tasks on geo-spatial and non-geo-spatial datasets as recommended in [17] using the introduced similarity functions, with the exception of the Latitude_Longitude similarity function (sim_{lat_lon}), which is unique to geo-spatial datasets.

Most research works in this field focuses on ways to enhance the accuracy of adaptive *k*NN methods and sometimes this comes at an extra cost in trading off execution time. This research work is tailored to cause a shift in paradigm towards the development of adaptive *k*NN methods that achieve faster classification without trading off accuracy. The results of the EACKER method based on the similarity functions introduced, demonstrated visible enhancement in the CPU running time compared to the ACKER algorithm. Furthermore, results show that there is no accuracy trade-off against the ACKER method, while improving the accuracy of the traditional *k*NN method for most of the classification tasks. The only dataset where the EACKER showed a poor performance alongside the ACKER and the traditional *k*NN is the SF - Crimes dataset. The probable reason the *k*NN and the adaptive *k*NN approach cannot achieve good performance with this dataset is the degree of class imbalance resulting from a skewed class distribution. Again, the experimental results of this research work show that using a large range of *k* is not essential, as after some point (k = 50), more *k* values do not improve classification accuracy. Therefore, reducing range of *k* values from *n* elements (where *n* could be the entire size of the dataset) to $\sqrt{n}$ elements doesn't result to a reduction in the overall classification accuracy, but can significantly decrease the running time.

Advancements to this research work can be achieved by investigating the influence of other similarity functions on the expected accuracy and the influence of other distance metrics on the overall accuracy. Secondly, the Simulated Annealing algorithm may be modified to search for optimal *k* adaptively by finding a steady set of nearest neighbors which will be dependent on the size and density distribution of any given dataset without having to search for k_{max}, as long as the overall classification accuracy is maintained.

References

[1] T. M. Cover, and P. E. Hart, "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory 13*, 1967, pp. 21–27.
[2] N. Nilsson, *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*, 1965.
[3] G. S. Sebestyen, *Decision-making Processes in Pattern Recognition (ACM monograph series)*, 1962.
[4] P. Rani, "A Review of Various KNN Techniques," *International Journal for Research in Applied Science & Engineering Technology*, 2017.
[5] J. Han, M. Kamber, and J. Pei, *Data Mining Concepts and Techniques.* Boston: Elsevier Inc, 2012.
[6] I. Tsang, J. Kwok, and P. Cheung, "Core Vector Machines: Fast SVM Training on Very Large Data Sets," *Journal of Machine Learning Research*, 2005, pp. 363-392.
[7] S. Zhou, T. Ling, and J. Guan, "Fast Text Classification: A Training Corpus Pruning Based Approach," *Proceedings of Database Systems for Advanced Applications*, Kyoto, Japan: IEEE Computer Society, 2003, pp. 127-136
[8] W. Ian, F. Eibe, "Data Mining: Practical Machine Learning Tools and Techniques," *Elsevier*, 2005, pp. 129-135
[9] S. Berchtold, B. Ertl, and Keim, "A Fast Nearest Neighbor Search in High-dimensional Space," *International Conference on Data Engineering*, 1998, pp. 209-218.
[10] B. LI, S. Yu, and Q. Lu, "An Improved K-Nearest Neighbor Algorithm for Text Categorization," in *Proceedings of the 20th International Conference on Computer Processing of Oriental Languages* . China: Shenyang, 2003.
[11] L. Jiang, H. Zhang, and Z. Cai, "Dynamic K-Nearest-Neighbor Naive Bayes with Attribute Weighted," *International*

Conference on Fuzzy Systems and Knowledge Discovery. Springer, Berlin, Heidelberg, 2006.
[12] C. Yang, Y. Li, C. Zhang, and Y. Hu, "*A Fast KNN Algorithm Based on Simulated Annealing".* Department of Computing & Information Technology, Fudan University, 2007.
[13] S. Sun, and R. Huang, "An Adaptive K-Nearest Neighbor Algorithm". *Seventh International Conference on Fuzzy Systems and Knowledge Discovery,* 2010.
[14] W. Liu, and S. Chawla, "Class Confidence Weighted KNN Algorithms for Imbalanced Datasets," *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2011, pp. 345-356.
[15] M. Kuhkan, "A Method to Improve the Accuracy of K-Nearest Neighbor Algorithm," *International Journal of Computer Engineering and Information Technology*, 2016, pp. 90-95.
[16] S. Mullick, S. Datta, and S. Das, "Adaptive Learning-Based K-Nearest Neighbour Classifiers With Resilience to Class Imbalance," *IEEE transactions on neural networks and learning systems*, 2018.
[17] M. Kibanov, M. Becker, J. Mueller, M. Atzmueller, A. Hotho, and G. Stumme, "Adaptive KNN Using Expected Accuracy for Classification of Geo-Spatial Data," *Symposium on Applied Computing.* Pau, France, 2018.
[18] M. Abdullahi, Md. Ngadi, "Hybrid Symbiotic Organisms Search Optimization Algorithm for Scheduling of Tasks on Cloud Computing Environment," *PLoSONE 11(6):e0158229,* 2016.

Authors' Profiles

Anozie Onyezewe was born on 6th August 1993. He received B.Sc. degree in Computer Science from Michael Okpara University of Agriculture Umudike and M.Sc in Computer Science from Ahmadu Bello University Zaria. In 2018 he joined the Ahmadu Bello University Department of Computer Science's Artificial Intelligence research group where he majored in Machine Learning and Intelligent Systems. In 2019, he was selected as a Student Instructor in Data Science Nigeria's AI+ Invasion Zaria.

Armand F. Donfack Kana received the B.Sc. degree in Computer Science from University of Ilorin, Nigeria, M.Sc. and Ph.D. degrees in Computer Science from University of Ibadan, Nigeria. He is currently a Senior Lecturer in Computer Science at the Department of Computer Science, Ahmadu Bello University, Zaria, Nigeria. His current research interests include Knowledge Representation and Reasoning, Machine Learning, Formal Ontologies and Soft Computing.

Aminu O. Abdulsalami is a lecturer in the Department of Computer Science, Ahmadu Bello University, Zaria. He received His B.Sc. and M.Sc. degrees in Computer Science from Ahmadu Bello University Zaria. Aminu is currently a Ph.D. student in the School of Computer Science and Technology, Wuhan University of Technology, China. His research interest includes Evolutionary Computing, Federated Learning and Blockchain.

Fatimah B. Abdullahi received her M.Sc. degree in Computer Science in 2009 from Ahmadu Bello University, Zaria Nigeria. Her Ph.D. degree in Computer Science from University of Liverpool United Kingdom in 2016. She is currently an academic staff member of Ahmadu Bello University. Her research interests include Data Mining, Machine Learning and Data Analytics.

How to cite this paper: Anozie Onyezewe, Armand F. Kana, Fatimah B. Abdullahi, Aminu O. Abdulsalami, "An Enhanced Adaptive k-Nearest Neighbor Classifier Using Simulated Annealing", International Journal of Intelligent Systems and Applications(IJISA), Vol.13, No.1, pp.34-44, 2021. DOI: 10.5815/ijisa.2021.01.03

