Using soP sets as an improvement for K-nearest Neighbors algorithm*

Norbert Luchowski[1,*,†], Piotr Chimiak[1,†]

[1]Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44100 Gliwice, POLAND

Abstract

In this research paper, we delve into the possibility of integration soft sets into the K-nearest neighbors (KNN) algorithm to enhance its performance, particularly in high-dimensional and large-scale datasets. Soft sets, known for their ability to handle uncertainty and vagueness in data, provide a robust framework that complements the traditional KNN method. Our study examines the efficiency of this hybrid approach across various datasets (which are based on MNIST, Modified National Institute of Standards and Technology database) differing in size and number of pixels. The results indicate a noticeable improvement in performance when applied to larger databases with higher dimensions, without compromising the accuracy observed in smaller datasets. Although the overall enhancement in performance is modest and does not surpass the accuracy achieved by well-optimized algorithms from existing libraries, the findings are promising. They suggest that soft sets offer a viable means to bolster the KNN algorithm, particularly in complex data scenarios. This research contributes to the ongoing efforts to refine machine learning techniques and highlights the potential of soft sets in achieving more efficient and accurate data classification. Further research is needed to optimize this approach and to explore its application in a broader range of machine learning tasks and datasets.

Keywords

MNIST dataset, KNN, Soft set,

1. Introduction

The k-nearest neighbor (KNN) algorithm is a non-parametric classification algorithm that assigns a test object to the decision class that is most common among its k nearest neighbors [3]. The classification of objects is based on the classes of the k nearest objects. KNN is popular and widely used so it is not strange that various modification and enhancements have been proposed, for instance [4, 2]. Soft sets, introduced by Molodtsov [6], offer a promising avenue for enhancing the KNN algorithm. Soft sets are capable of dealing with uncertainties and ambiguities, making them particularly suited for complex data environments. Soft sets provide a flexible mathematical framework that can be leveraged to improve the robustness and adaptability of the KNN algorithm, potentially leading to better performance [8, 7]. Despite the potential benefits, the integration of soft sets with KNN has not been extensively explored, the application of soft sets to enhance core machine learning algorithms like KNN remains an under researched area. In this context, our paper aim to fill the gap by evaluating the performance of KNN algorithm enhanced with soft set. We also examine the results of the algorithm with a decision based on probability.

*IVUS2024: Information Society and University Studies 2024, May 17, Kaunas, Lithuania

1,* Corresponding author
† These author contributed equally.
nl307893@student.polsl.pl (N. Luchowski); pc307850@student.polsl.pl (P. Chimiak)
0009-0004-4475-6720 (N. Luchowski); 0009-0002-0532-2474 (P. Chimiak)

Our findings contribute to the efforts to refine and improve machine learning techniques, offering insights that could lead to more efficient and accurate data classification methods. Further research in this area is essential to optimize the approach and explore its broader applications in machine learning.

Algorithm 1 K-Nearest Neighbors (KNN) Algorithm

Require:
$\mathbf{X}$: Training data features
$\mathbf{y}$: Training data labels
$\mathbf{x}_{\mathbf{new}}$: New data point
k: Number of neighbors
Ensure:
Predicted label for $\mathbf{x}_{\text{new}}$

function KNN($\mathbf{X}$, $\mathbf{y}$, $\mathbf{x}_{\mathbf{new}}$, k)
  distances $\longleftarrow$ []
  for $i \leftarrow 1$ to n **do**
    $d \leftarrow$ EuclideanDistance($\mathbf{X}[i]$, $\mathbf{x}_{\text{new}}$)
    distances.append((d, $\mathbf{y}[i]$))
  end for
  distances $\longleftarrow$ Sort(distances)
  neighbors $\longleftarrow$ distances[1:k]
  labels $\longleftarrow$ [label for (dist, label) in neighbors]
  prediction $\longleftarrow$ Mode(labels)
  return prediction
end function
function EuclideanDistance($\mathbf{x}_1$, $\mathbf{x}_2$)
  return $\sqrt{\sum_{j=1}^{m} (\mathbf{x}_1[j] - \mathbf{x}_2[j])^2}$
end function

2. Methodology

Our methodology encompasses a systematic approach to evaluate the effectiveness of the soft set-enhanced K-nearest neighbors (KNN) algorithm across a diverse range of datasets. The methodology is designed to provide rigorous experimentation and analysis, ensuring robust conclusions regarding the performance of the proposed approach. To ensure the accuracy of the research, we carried out several measurements on a differently divided and shuffied datasets. We also use in analysis confusion matrix where each row is an actual class while each column is in a predicted class. In addition to the overall accuracy, we also examine the precision recall, f1score for individual classes, digits. The k-Nearest Neighbors begins with choosing the number of neighbors, typically a small odd number like 3 or 5, we chose 3. For a new data point, the algorithm calculates the distance between this point and all points in the training dataset,for that we used euclidean distance but it can be changed. The k closest points (neighbors) are identified. For classification, the most common class among these neighbors is assigned to the new data point if there is a draw the result is picked randomly from nearest neighbors. Our modified

Algorithm 2 Soft Set Based Prediction Algorithm

```
Require:
    X_train : Training data features
    y_train : Training data labels X_test
    : Test data features
    func : Function to compute soft set elements
Ensure:
    Predicted labels for X_test
    Step 1: Create Soft Set
    function CreateSofSet(X_train, y_train)
        soft_set ← []
        mean ← Mean(X_train, axis=0)
        for each y in Unique(y_train) do
            X_y ← X_train[y_train == y]
            mean_y ← Mean(X_y, axis=0)
            soft_set.append(mean_y)
        end for
        soft_set ← Array(soft_set)
        return soft_set
    end function

        Step 2: Prediction
    function Predict(X_test, soft_set) prediction
        ← []
        for each x in X_test do
            scores ← DotProduct(soft_set, x)
            result ← ArgMax(scores)
            prediction.append(result)
        end for
        return prediction
    end function

    Main Execution
    soft_set ← CreateSofSet(X_train, y_train)
    predictions ← Predict(X_test, soft_set)
    function Argsort(array)
        return Indices that would sort the array
    end function
    function DotProduct(array1, array2)
        return Dot product of array1 and array2
    end function
    function ArgMax(array)
        return Index of the maximum value in array
    end function
    function Mode(array)
        return Most frequent element in array
    end function
    function Length(array)
        return Length of array
    end function
```

Algorithm 3 Enhanced K-Nearest Neighbors (KNN) with Soft Set

Class Definition: KNN_Soft_Set

```
function Fit(self, X_train, y_train)
    self.X_train ← X_train
    self.y_train ← y_train
    self.soft_set ← []
    mean ← Mean(X_train, axis=0)
    self.soft_set ← CreateSoftSet(self.X_train, self.y_train)
end function
function predict(self, X_test)
    predictions ← []
    for each x in X_test do
        distances ← [EuclideanDistance(x, x1) for x1 in self.X_train]
        indices ← Argsort(distances)[:self.k]
        labels ← [self.y_train[i] for i in indices]
        if Length(unique_labels) == self.k then
            soft_set_subset ← self.soft_set[unique_labels]
            scores ← DotProduct(soft_set_subset, x)
            most_common ← ArgMax(scores)
            predictions.append(unique_labels[most_common])
        else
            result ← Mode(labels)
            predictions.append(result)
        end if
    end for
    return predictions
end function
```

KNN works the same but when draw occurs the best promising result from soft set evaluation is being picked. The prediction model using soft sets involves creating a representative vector for each class based on the mean of feature vectors. These vectors are used to classify new data points by projecting them into the space defined by these representative vectors. The class of the new data point is determined by the highest projection value. We also tried with vectors in binary space using thresholds which are the boundary between 0 and 1 value, but such created prediction model were a little less accurate than simple one using only means that was forsaken during experiments phase.

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2}$$

TP - true positive, TN - true negative, FP - false positive, FN - false negative

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

Algorithm 4 Enhanced K-Nearest Neighbors (KNN) with Soft Set and probability

Class Definition: KNN_Soft_Set

function FIt(*self*, $\mathbf{X}_{\text{train}}$, $\mathbf{y}_{\text{train}}$)
    self.X_{train} ← $\mathbf{X}_{\text{train}}$
    self.y_{train} ← $\mathbf{y}_{\text{train}}$
    self.*soft_set* ← []
    mean ← MEAn($\mathbf{X}_{\text{train}}$, axis=0)
    self.*soft_set* ← CrEAtESoFtSEt(*self*.X_{train}, *self*.y_{train})
end function
function prEdIct(*self*, $\mathbf{X}_{\text{test}}$)
    predictions ← []
    for each x in $\mathbf{X}_{\text{test}}$ **do**
        distances ← [EUcLIdEAnDIstAncE(x, $x1$) for $x1$ in *self*.X_{train}]
        indices ← Argsort(*distances*)[:*self*.*k*]
        labels ← [*self*.$y_{\text{train}}[i]$ for i in *indices*]
        if LEngth(*unique_labels*) == *self*.*k* **then**
            soft_values ← *self*.*soft_set*[*unique_labels*]
            soft_scores ← DotProdUct(soft_values, x)
            probabilities ← soft_scores / SUm(soft_scores)
            chosen_label ← RAndomChoIcE(unique_labels, p=probabilities)
            Append *chosen_label* to *prediction*
        else
            result ← ModE(labels)
            predictions.append(result)
        end if
    end for
    return predictions
end function
function RAndomChoIcE(labels, probabilities)
    r ← random number between 0 and 1
    cumulative_probability ← 0
    for i from 0 to LEngth(labels) **do**
        cumulative_probability ← *cumulative_probability* + *probabilities*[i]
        if $r \leq$ *cumulative_probability* **then**
            return *labels*[i]
        end if
    end for
end function

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

Prediction for soft set: for each class C_i, calculate the mean vector μ_i in set A:

$$\mu_i = \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j \tag{5}$$

where $|C_i|$ is the number of vectors in class C_i. We can also use binary format using a given threshold t

$$\hat{\mu}_i = \begin{cases} 1, & \text{if } \mu_i \geq t \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where $\hat{\mu}_i$ represents the binary output of μ_i. Create a matrix M whose columns are the transposed mean vectors μ_i^T:

$$M = [\mu_1^T \; \mu_2^T \; \dots \; \mu_k^T]$$

where k is the number of classes.

For a given test vector **t**, multiply **t** by the model matrix M:

$$\mathbf{v} = \mathbf{t} \cdot M$$

where **v** contains the projection values of **t** onto each class mean vector.

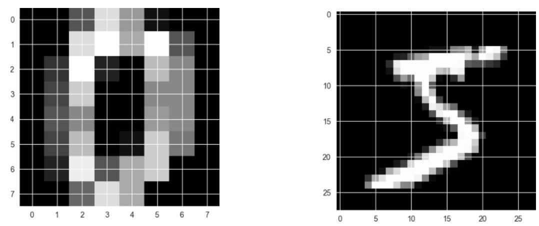
Identify the index i of the maximum value in **v**.

The predicted class for the test vector **t** is the class corresponding to this index.
For each class C_i in set A, the probability $P(C_i)$ is calculated based on the projection vector **v** as follows:

$$P(C_i) = \frac{v_i}{\sum_{j=1}^{k} v_j} \tag{7}$$

where v_i is the i-th index of the vector **v**, and k is the number of classes in set A. The last equation is used in fourth algorithm.

Figure 1 Comparison of example images of different sizes



(a) Example 8x8 image

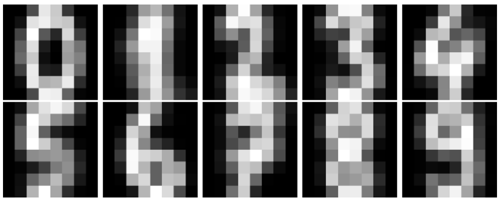
(b) Example 28x28 image

3. Experiments

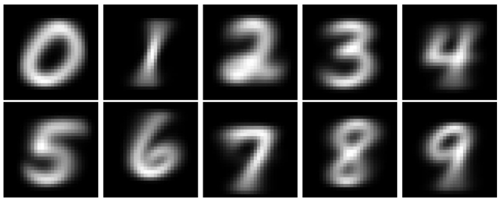
In order to validate hypothetical improvement of the effectiveness of our proposed enchanced KNN with soft set algorithm we conducted a series of calculation using 2 varioius datasets based on MNIST database. The purpose of these experiments was to assess the algorithm's performance in different data scenarios, including different datasets varying the number of dimensions and databse size. Through thse experiments, we aim to validate our hypothesis. The first database we used sklearn copy of [1] that contains 1797 samples where data-point is an 8x8 image of a digit which gives us 64 dimensions. Pixels are describe as integers between 0 and 16. The second is [5] database which contains 60,000 records as a train set and 10,000 records as a test set which gives us total 70,000 28x28 images (784 dimensions) where pixel values range from 0 to 255.

First of all, we created a template representation of digits, using mean values of all pixel. We have not used threshold for that cause accuracies achieved by soft set prediction models were worse in smaller dataset by 0.16, in bigger difference is insignificant.

Figure 2 Comparison of soft set representations for different image sizes

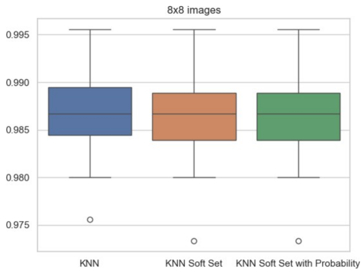


(a) Soft set representation 8x8 images

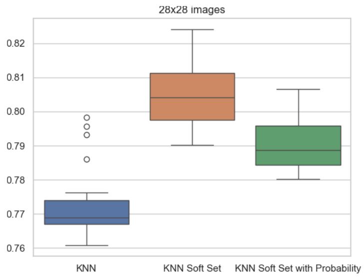


(b) Soft set representation 28x28 images

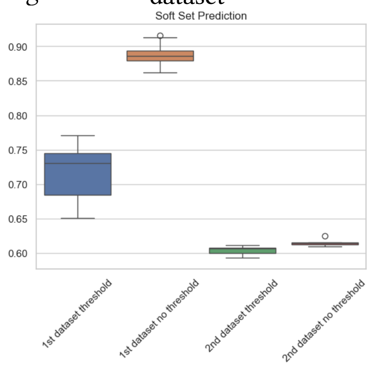
Figure 3 Comparison of achieved accuracies for different datasets and algorithms



(a) Achieved accuracies for different algorithms, 8x8 images dataset



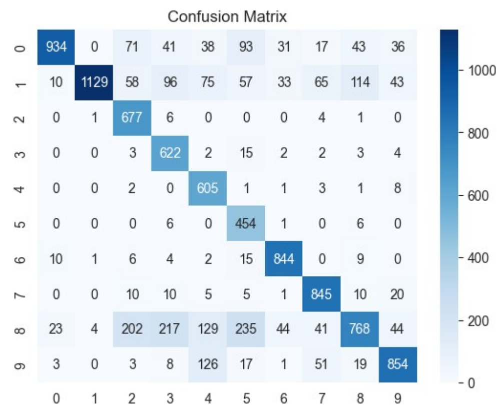
(b) Achieved accuracies for different algorithms, 28x28 images dataset



(c) Achieved accuracies for soft sets

Using all three algorithms (KNN, KNN with soft set, KNN with soft set and probability) on 1st database we obtained accuracy around 0.986, we have not seen an improvement and even minor reduction while using enhanced KNN However on 2nd database we have seen an improvement, accuracy for KNN: 0.773, accuracy for enhanced KNN 0.805. Enhanced KNN with probability was better than normal KNN, but worse by 0.13 in comparison with enhanced KNN. Therefore we can see an improvement in large dataset by around 4

Figure 4 Comparison of KNN confusion matrices for the second dataset



(a) Confusion matrix, the second dataset, KNN

(b) Confusion matrix, the second dataset, KNN enhanced with soft set

Table 1 Precision, recall i F1 score 2nd dataset, KNN

	Precision	Recall	F1 Score
0	0.95306122	0.7162576687116564	0.8178633975481612
1	0.99471366	0.6720238095238096	0.8021314387211368
2	0.65600775	0.9825834542815675	0.7867518884369552
3	0.61584158	0.9525267993874426	0.7480457005411906
4	0.61608961	0.9742351046698873	0.7548346849656893
5	0.50896861	0.9721627408993576	0.6681383370125092
6	0.88100209	0.9472502805836139	0.9129259058950784
7	0.82198444	0.9326710816777042	0.8738366080661841
8	0.78850103	0.44991212653778556	0.5729205520328235
9	0.84638256	0.7892791127541159	0.8168340506934482

Table 2 Precision, recall i F1 score 2nd dataset, enhanced KNN

	Precision	Recall	F1 Score
0	0.95816327	0.765905383360522	0.8513145965548504
1	0.99471366	0.6909424724602203	0.8154568436258577
2	0.73546512	0.9768339768339769	0.8391376451077943
3	0.71683168	0.9061326658322904	0.8004422332780542
4	0.67718941	0.9540889526542324	0.7921381774865992
5	0.51793722	0.9705882352941176	0.6754385964912281
6	0.90814196	0.925531914893617	0.916754478398314
7	0.84046693	0.9310344827586207	0.8834355828220859
8	0.79158111	0.5530846484935438	0.6511824324324325
9	0.86917740	0.7767936226749336	0.8203928905519177

Table 3 Precision, recall i F1 score 1st dataset, KNN

	Precision	Recall	F1 Score
0	1.00000000	1.0	1.0
1	1.00000000	0.8936170212765957	0.9438202247191011
2	1.00000000	1.0	1.0
3	1.00000000	0.9523809523809523	0.975609756097561
4	1.00000000	0.9811320754716981	0.9904761904761905
5	0.95833333	1.0	0.9787234042553191
6	1.00000000	1.0	1.0
7	1.00000000	1.0	1.0
8	0.91111111	1.0	0.9534883720930233
9	0.90476190	0.95	0.926829268292683

Table 4 Precision, recall i F1 score 1st dataset, enhanced KNN

	Precision	Recall	F1 Score
0	1.00000000	1.0	1.0
1	1.00000000	0.8936170212765957	0.9438202247191011
2	0.97560976	1.0	0.9876543209876543
3	1.00000000	0.9523809523809523	0.975609756097561
4	1.00000000	0.9811320754716981	0.9904761904761905
5	0.95833333	1.0	0.9787234042553191
6	1.00000000	1.0	1.0
7	1.00000000	0.9761904761904762	0.9879518072289156
8	0.91111111	1.0	0.9534883720930233
9	0.90476190	0.95	0.926829268292683

4. Conclusion

Our research demonstrates that hybrid approach and integrating soft sets into the K-nearest neighbors algorithm yields 4% increase in accuracy when applied to higher-dimensional and large dataset [5]. The enhancement does not compromise the accuracy in smaller databes [1], maintaining almost the same performance levels with traditional KNN. While this improvement is noteworthy, we know the our algorithm which performed the calculation is not even decent and does not surpass the accuracies achieved by well-optimized algorithms available in existing libraries. Despite these modest gains, the findings are promising and indicate the potential of soft sets to enhance KNN performance. The research underscores the need for further investigation to optimize this approach and determine whether it can eventually outperform the currently in use implementation of KNN. Future work should focus on refining the algorithm, particularly by exploring new ways of constructing soft sets, to further boost accuracy and efficiency and also improving the accuracy of KNN in bigger datasets by using much more sophisticated implementation than we used in our research. Additionally, expanding the scope of testing to include a more diverse range of datasets and real-world applications could provide deeper insights into the strengths and limitations of this approach whether the boost is only achievable in digit recognition or in more wider spectrum.

References

[1] E. Alpaydin and C. Kaynak. *Optical Recognition of Handwritten Digits*. This dataset is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license. 2019. doi: 10.24432/C50P49. url: https://archive.ics.uci.edu/dataset/80/optical+recognition+of+handwritten+digits.

[2] Guo-Feng Fan et al. *Application of the Weighted K-Nearest Neighbor Algorithm for Short-Term Load Forecasting*. 2019. doi: 10.3390/en12050916. url: https://doi.org/10.3390/en12050916.

[3] Richard Jensen and Chris Cornelis. "A New Approach to Fuzzy-Rough Nearest Neighbour Classification". In: *Lecture Notes in Computer Science*. Vol. 5306. Springer, 2008.

[4] James M. Keller, Michael R. Gray, and James A. Givens. "A fuzzy K-nearest neighbor algorithm". In: *IEEE Transactions on Systems, Man, and Cybernetics* (1985). doi: 10.1109/TSMC.1985.6313426.

[5] Yann LeCun and Corinna Cortes. *The MNIST Database of Handwritten Digits*. MNIST dataset is a derivative work from original NIST datasets. It is made available under the terms of the Creative Commons Attribution-Share Alike 3.0 license. 2010. url: http://yann.lecun.com/exdb/mnist.

[6] D. Molodtsov. "Soft set theory—First results". In: *Computers & Mathematics with Applications* 37.4-5 (1999), pp. 19–31. doi: 10.1016/S0898-1221(99)00056-5.

[7] Marcin Woźniak and Dawid Połap. "Object detection and recognition via clustered features". In: *Neurocomputing* 320 (2018), pp. 76–84.

[8] Marcin Woźniak and Dawid Połap. "Soft trees with neural components as image-processing technique for archeological excavations". In: *Personal and Ubiquitous Computing* 24.3 (2020), pp. 363–375.