Implementation of KNN Classifier for Sequence Prediction

Zaka Ahmed
zaka.ahmed@stud.fra-uas.de

Muhammad Haris
Muhammad.Haris@fra-uas.de

***Abstract*—Machine learning comes out as an advanced and significant technology, evolving in our daily lives. The focus of Machine Learning is to enable computers to learn from the data and make predictions or decisions based on that data, enabling informed decision-making without the need for extra instruction or explicit programming for the system to perform tasks better. The sectors in which machine learning is involved include education, telecommunication, retail, research and development, finance, healthcare, and transportation through data-driven insights. Machine learning has three basic types: Supervised Learning, Unsupervised Learning, and Reinforcement learning. In the domain of supervised machine learning, a variety of classifiers abound, including decision trees, support vector machines, Naive Bayes, and K-nearest neighbors (KNN), each tailored to address specific data analysis challenges. Among the numerous machine learning algorithms, the K-Nearest Neighbors (KNN) classifier is the simplest and most effective for classification problems. This paper presents the implementation of the k- k-nearest neighbor (KNN) machine learning algorithm for predicting outcome variables based on input variables. Leveraging the capabilities of Hierarchical Temporal Memory (HTM) for learning complex temporal patterns, our study focuses on predicting types of sequences: even number sequences, odd number sequences, and decimal number sequences. We integrate the KNN model with the Neocortex API to efficiently classify sequences. The KNN model receives input data in the form of Sparse Distributed Representations (SDR) from HTM. We construct a dataset comprising multiple sequence SDRs, each with varying values within a defined threshold. The KNN model processes a stream of sequence SDRs, with the dataset split into 70% training data and 30% testing data. During testing, the model accurately classifies sequences, achieving a 90.9% accuracy rate with some SDR testing data, and consistently predicts matches with 100% accuracy in most cases. The paper discusses the KNN design procedure, challenges encountered, and potential enhancements to further improve model accuracy.**

***Keywords*—** *Machine Learning, Hierarchical Temporal Memory, K-Nearest Neighbors, Sequence Classification, Integration, Neocortex API, Accuracy Enhancement.*

I. Introduction

KNN was first developed by Joseph Hodges and Evelyn Fix in the year 1951 where further development and we can say modification were led by Thomas Cover's [1] statistics the concept of the k-nearest neighbors algorithm(KNN) is involved in the non-parametric supervised learning method. KNN is commonly used for regression and classification. The input consists of K-neighbor training examples in both regression and classification. Remember the output depends on whether the use case is either regression or classification of K-NN.

A. Regression

The main difference between classification and regression is that in regression, the output that we get will become the object's property value. The value is the sum of the nearest neighbor's values averaged together. If k=1, the output is assigned from that single nearest neighbor.

B. Classification

The primary difference between the classifier and regression outputs is that in the former case, the output is the class membership. In classification, the object is classified based on the votes of its nearest neighbors. If k = 1, then the object will be in the class of that single nearest neighbor. The simple function of the KNN model is to predict the target class label. In other words, the class label is often described as a majority voting. The most common terms are technically considered" plurality voting" and" majority vote" The term" majority voting" means the majority needs to be greater the 50% for making decisions. The classification problems with only two classes, like binary predictions, there is always a majority. A majority vote is also automatically a plurality vote. We don't require multi-class settings to make predictions via KNN in multi-class settings. [2]

II. Methods

This section includes multiple subsections including a description of KNN in detail, The second section tries to focus on the Theoretical background including how the values determined the accuracy and complexity of the classifier. The last section of this section is regarding Hierarchical Temporal Memory (HTM).

A. Literature Review

In the Literature, we have discussed various methods which are used by different authors. K-Nearest Neighbors (KNN) algorithm is a widely used non-parametric classification method in machine learning. KNN relies on instance-based learning, in which the similarity of newly added data points to previously labeled data points is used to make predictions.

In finance, KNN is used for credit scoring, stock analysis, and fraud detection. In the healthcare sector, KNN is used for patient health monitoring and drug suggestions. If I talk about marketing professionally KNN is used for product suggestion, market trend analysis, and customer ad suggestions. In most cases, KNN is used for determining specific patterns and tasks like face recognition or image classification. Researchers have proposed various enhancements to the traditional KNN algorithm which can improve scalability and performance.

The weight adjustment algorithm proposed by Han EH. proposed assigning weights to the nearest neighbors based on their distance from the respective point [3]. The assigned weights distinguish, how much the weights influence the classification method. In this way, high weights will be assigned to the ones who are closer neighbors, so it gives more priority to similar instances while performing classification. This technique is useful where the dataset has many features, some of which can be considered unnecessary, but it has a high cost in the context of computational cost.

Zhang et al. have proposed to adjust the value of K based on the local density of data points [4]. This adaptive KNN algorithm dynamically selects the optimal value of K for each query point, leading to more robust predictions. And shows that the adaptive algorithm outperforms many other traditional KNN algorithms. The other approach can be a locally adopted KNN algorithm. It chooses the optimal value of K for classifying an input by analyzing the outcomes of cross-validation computations within the local neighborhood of the unlabeled data point. [5] The approach defined by Song Yang [5] is to introduce two input parameters. As we know determining the correct value of K depends on the characteristics of the dataset, the selection of the correct parameter for various applications is a challenge. Song Yang suggests introducing a novel metric that assesses the informativeness of objects to be classified, with informativeness quantifying the significance of data points. Two parameters will be K and I as input. The class is determined based on the majority class of the most informative training examples. Whereas some have proposed to integrate KNN with dimensionality reduction methods such as principal component analysis(PCA) to improve computational efficiency [6]. This combination is an example of a hybrid approach. Whereas others have combined KNN with ensemble methods such as random forests to enhance predictive accuracy. These hybrid approaches enhance the performance as compared to standalone KNN.

B. K-Nearest Neighbors Parameters and Matrix

The K-nearest was first used by the US Air Force to execute characteristics analysis. Different parameters in the KNN classifier play an important role in algorithm designs including distance matrix, K-Value selection, and voting.

1) Computing Distance Matrix

If we try to revise, the k-nearest neighbor algorithm's main objective is the identification of the nearest neighbors around the input. The step will be given a label or name class to that specific point. The first thing is determining the distance metrics. To find which class (data point) is nearest to the input data, to do so we will calculate the distance between the data points and query point. We get assistance to decide in which regions the input point belongs. The distance metrics can be either Manhattan distance or any other approach. The first thing is to identify the k-nearest neighbors and then the number of its k-neatest neighbors. The most famous techniques are discussed below:

a) Euclidean distance (p=2)

In the early 300s Era before, mathematicians of Greece introduced Euclid while finding the difference between distance and angle. Still, Euclid is the most commonly used of distance. From that early, till yet Euclid is widely and applies in two- or three-dimensions space. [7] The main method is the root of square distances between two coordinates of a pair of objects. Then there is a square root of the sum of squares of the differences between the corresponding values.

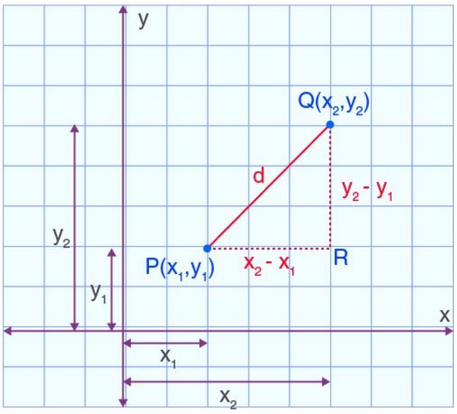


Fig 1: Euclidean Distance [7]

As from Fig. 1, we have the (x_1, y_1), and (x_2, y_2), we can see that (x_1, x_2) , and (y_1, y_2) are allocated into the two-dimensional space. If we try to build a right-angled triangle and draw a hypotenuse straight line between two points(d). The other two sides of a right-angle triangle are base and altitude which will be $|x^1 - y^1|$ and $|x^2 - y^2|$. So there the hypotenuse(d) is our Euclidean distance which is between (x_1, x_2),, and (y_1, y_2) . As this is only a straight line we will use Pythagorean theorem. The distance between (x_1, x_2),, and (y_1, y_2), would be $(x_1, y_1)^2$, and $(x_2, y_2)^2$.

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

- yi are the coordinates of one point.
- xi are the coordinates of another point.
- d is the distance between $(x_1, y_1), and, (x_2, y_2)$.

Moreover, to find the Euclidean distance between, y_n, and x_n the Euclidean distance is $(x_n - y_n)^2$ in n-dimensional space. Euclidean is limited to real-valued vectors and is most used for distance measures. The straight line between the input point and the other point under measurement is found using the expression above.

Euclidean distance is the shortest distance between two points means the straight line between the source and destination whereas the Manhattan distance is the sum of all the real distances between the source and destination. And each always be a straight line. The reason behind Euclidean distance used in KNN is that it provides a simple and intuitive measure of similarity between data points in a multi-dimensional space. One of the main reasons is that

Euclidean distance in KNN is its computational efficiency and simplicity. [8] Euclidean distance metric works well for continuous features and is suitable for data represented in a high-dimensional space.

Euclidean Distance is largely used in machine learning learning algorithms such as K-Nearest(KNN) where it is used to measure the similarity between two data points. Spatial Analysis for measuring distance between geographical locations. [8] As well as, in robotics for obstacle avoidance and path planning. It is also used in the image process for comparing images based on pixel values. Due to the coordinate system, it is robust to transformations of the data. It is the most widely adopted approach in machine learning applications.

b) Manhattan distance

In Manhattan distance, the absolute value is measured between two points. Manhattan is widely used for resolving problems related to geometry. We can say it is an ordinary distance between two points. Manhattan distance is also one of the popular and dominant distance metrics. The most common example is the Uber app visualization with a grid and navigation via streets from one address to another illustration. Manhattan is widely used in cluster analysis. The k-Means clustering algorithm is the common example where the Manhattan distance is used. [9] The second name of Manhattan Distance is city Block Distance. The easiest method to figure out the distance is to move from one point to the other between two locations by moving horizontally and then vertically, rather than straight forward. [10] It only needs to subtract instead of performing complicated calculations that why it is one of the simplest methods of calculating the distance.

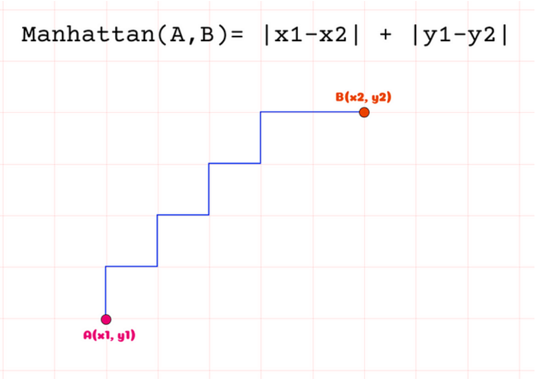


Figure 2: Manhattan Distance [10]

Manhattan Distance = d(x, y) = $\sqrt{(\sum_{i=1}^{m} |x_i - y_i|)}$

As shown[56], it is the root of the squared difference between the coordinates of two objects. We only need to subtract two points instead of performing complex tasks.

c) Minkowski Distance

It is the extrapolated form of Manhattan and Euclidean distance matrices. Manhattan distance is denoted by p equal to one whereas the Euclidean distance is represented by p equal to two [11]. Parameter p allows the creation of other distance metrics as shown below.

Minkowski Distance = d(x, y) = $\sqrt{(\sum_{i=1}^{n} |x_i - y_i|)^{1/p}}$

d) Hamming Distance

This technique is used with string vectors or Boolean identifying the points where the vectors do not match. [12] Overlap metrics are also referred to as represented below:

$$Hamming\ Distance = D_H = (\sum_{i=1}^{k} |x_i - y_i|)$$

In simple words, k is the number of neighbors used for making predictions. The k value indicates how many neighbors will be compared or we can say checked to determine the resultant class in the KNN Algorithm. For example: By changing the value of k the classification can lead to under-fitting or over-fitting. If k=1, the instance will be assigned to the same class because we have a single neighbor. Using a lower value of k can have low bias, but high variance. As well as a larger value of k may lead to lower variance and high bias. [13] So, we can define k as the balancing act as different values impact the variance on under-fitting or overfitting. To avoid ties in classification, k is recommended to be an odd number. The best approach to get an optimal k for your dataset is cross-validation tactics.

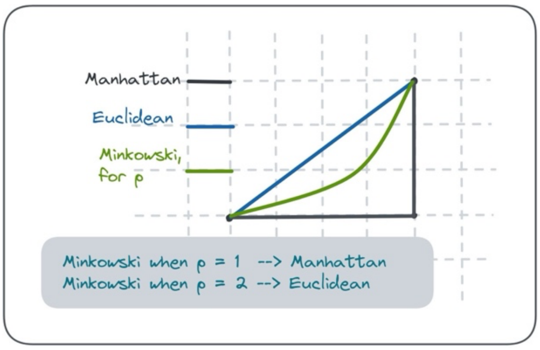

Manhattan
Euclidean
Minkowski, for p
Minkowski when p = 1 --> Manhattan
Minkowski when p = 2 --> Euclidean

Figure 3: Difference Between Distance Methods [13]

2) Defining K Selection

In simple words, k is the number of neighbors used for making predictions. The k value indicates how many neighbors will be compared or we can say checked to determine the resultant class in the KNN Algorithm. For-example: By changing the value of k the classification can lead to under-fitting or over-fitting. If k=1, the instance will be assigned to the same class because we have a single neighbor. Using the lower value of k can low bias, but high variance.

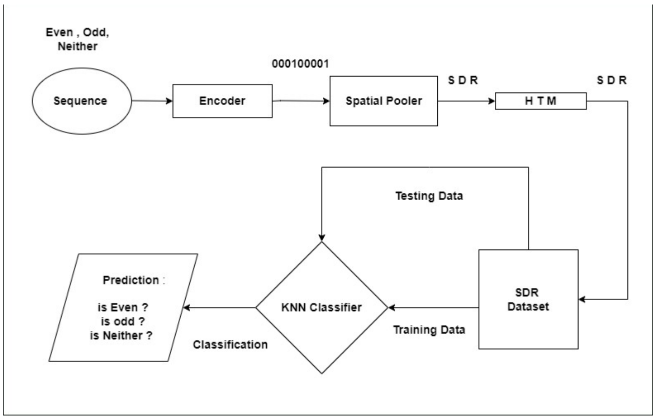


Figure 4: Process Representation

As well a larger value of k may lead to lower variance and high bias. [14] So, we can define k as the balancing act as different values impact the variance on under-fitting or over-fitting. To avoid ties in classification, k is recommended to be an odd number. The best approach to get an optimal k for your dataset is cross-validation tactics.

In the KNN algorithm, the prediction is based on the number of K nearest neighbors considered. The predicted or winning class is determined using the majority voting principle, the class that has high numbers of votes is the class from which our element belongs. The highest frequency among the K nearest neighbors is chosen. However, the important thing is that it is essential to recognize that the choice of K can impact the voting outcome. The higher value of K may lead to less confident predictions. [2] [15] [3] The scenarios where we have imbalanced data, where one class is higher or outweighs the other class, the majority voting principle can be biased. To resolve such kind of issues techniques like distance-weighted voting and weighted voting us introduced which can help to balance the influence of each class.

Mathematically, if $(C_1, C_2, C_3, \ldots. C_n)$ represent the unique class labels among the K nearest neighbors, and $Count(C_i)$ represents the count of occurrences of class the predicted class (y) for the new observation is determined by:

$$y = argmaxCount(Ci)$$

Where y is the predicted class Label. This voting equation ensures that the predicted class is the one that is most prevalent among the K nearest neighbors. Techniques like normalization, standardization, and scaling play a crucial role in preparing the data for modeling in pre-processing. [16] Normalization involves scaling the features to a similar range, typically between 0 and 1, the main advantage of normalization is that it prevents larger scales from dominating the model. Scaling adjusts the range of features to a desired range, which can enhance the performance of algorithms sensitive to feature magnitudes.

As the process diagram (Fig:4) shows the step-by-step procedure, we have adopted in the implementation. In the first step, we feed the input sequences from the JSON file (3.1) and that specific sequence gets encoded. Then after spatial pooler, we get sequence SDRs. Then it will be stored in HTM. The next step will be the implementation of a KNN classifier in which the test data will be mapped, the KNN helps to predict the class from where the sequence blogs to that specific class. The sudo-code of voting is also discussed in the implementation part of the paper.

3) Overview of HTM

The objective behind the HTM CLA is to make progress towards making a program that functions cognitive tasks like a simple human being brain. The prediction is done by making a system that can memorize as well as learn from the information executions that were fed before. HTM to anticipate and memorize, requires user Input.

The overall HTM has multiple sections which include data, Encoder, HTM spatial Pooler, HTM temporal Memory, and HTM Classifier [17]. The data which is also known as input is a scalar value, date or time, or a picture. Then the next element is the encoder which is responsible for changing the data into SDR which can further be used with the HTM classifier. SDR is in the cluster of binary input either '0' or '1'. As discussed earlier, the input of the encoder can be anything like a scalar value. It includes locations, weeks, months, times, or days in a week, etc.

The next part of the HTM is a spatial pooler, which is an algorithm that learns spatial patterns. The spatial pooler gets an input of bits cluster and converts it into SDR. The next Part is Temporal memory, a part which learns the arrangements of SDRs shaped by the spatial pooler algorithm. [18]

Spatial Poolar is the second phase of HTM learning. It uses the output of the encoder to learn the SDR of the given input binary array. The idea for the spatial poolar SDR is to generate SDRs of the input which is the output of the encoder. [19] Once the input SDR are learned, if the same input is given again, it tries to match already learned SDRs and then generates a similar matching SDR. In this method, it will disgunish, if is it the same input that is already memorized or a different one.

4) Spatial Pooler

Spatial Poolar is the second phase of HTM learning. It uses the output of the encoder to learn the SDR of the given input binary array. The idea for the spatial poolar SDR is to generate SDRs of the input which is the output of the encoder. [17] Once the input SDR are learned, if the same input is given again, it tries to match already learned SDRs and then generates a similar matching SDR. In this method, it will disguise, whether is it the same input that is already memorized or a different one.

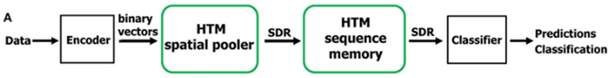


Figure 5: Spatial Poolar [51]

III. Implementation Details

The KNN method is divided into two classes first and the main class is the classifier which consists of distance, vote, and classify. The second class for indexanddistance. Distance method which is used to calculate the distance of a new data point that has unlabeled data. The voting method is used to prepare the voting table from the distance matrix and the Classify method is used to classify the class of the unknown label data as an output. Remember we are using text files as our dataset which is slit into two parts training and test dataset. These details are discussed in the further section in details.

A. Training and Test Dataset

In the approach, We have tried to give hardcore arrays as a training and testing Dataset. Then, we decided to use text files instead of hardcore values. The last index is for the identification of the class from which the data belongs. Later after trying various other methods. We have decided to go with initializing a List for the data matrix and save it in a JSON File. We have divided the file in such a way that 70% of the data acts as training data and the other 30% of data in the file format which acts as a test dataset. The large data sets, the technique is not scaled able, because the k-NN classifier stores all the training data in memory. Below is an example format of our JSON file data.

```
[{
"SequenceName": "S1",
"SequenceData": [8039, 8738, 9334,.......,11124]
},
{
"SequenceName": "S2",
"SequenceData": [8191, 8913, 9405,.......,11419]
},
{
"SequenceName": "S3",
"SequenceData": [9724, 9831,10092, .......,12771]
}]
```

Table 1: JSON File & Dataset

We have saved the label and training points. As Sequence-Name is the class name to which this dataset belongs. We have 3 types of classes: the first class is even numbers, the second class is odd numbers elements and the third class is decimal class which is also known as neither odd nor even. The even class is represented by S1, the odd class is represented by S2 and the decimal class dataset is represented by S3 as shown in the above example.

```
FUNCTION SplitDataset(sequenceDataList, trainingFeatures,
   trainingLabels, testingFeatures, testingLabels, Splitingratio)
   :
INITIALIZE trainingFeatures TO List<List< double>>()
INITIALIZE trainingLabels TO List<string>()
INITIALIZE testingFeatures TO List<List<double>>()
INITIALIZE testingLabels TO List<string>()

CREATE rand Random object

FOR EACH entry IN sequenceDataList DO
SET label TO entry.SequenceName
SET features TO entry.SequenceData

IF rand.NextDouble() < Splitingratio THEN
   ADD a copy of features to trainingFeatures
      ADD label to trainingLabels
ELSE
   ADD a copy of features to testingFeatures
      ADD label to testingLabels
END IF
END FOR
END FUNCTION
```

The above code is used for splitting the dataset. As we have already discussed the label of class can be either S1, S2, or S3. For splitting our data this method is used. Test data, which is 30% of our file, is used to determine the accuracy of our implementation. The classifier function accepts multiple parameters from the item to predict, it includes the number of classes, training data, and testing data which are also named testing features, training features, and training labels. We have 3 classes, the number of nearest neighbors to evaluate, and the matrix of training data, so the classifier is the function that determines or classifies our data. Further discussed in the KNN implementation section.

B. Approaches and Challenges

In this, we have discussed various approaches that we have used throughout the project.

1. Classification of numbers from 0 to 9 in a sequence based on Sparse Distributed Representations (SDRs)

At the start, after investigating the KNN Classifier and how it runs on C#, we develop a KNN Algorithm evaluation using random unclassified features and known features with label classes Then we explore the Neocortex repository which we fork, understand how it generates SDR's based on the sequence, and how can we used to classify the value in the sequence based on these SDR's Value. At the start we gave a sequence of 0 to 9 to sequence the learning experiment and extract the SDRs of sequence 0 to 9 then we labeled the SDRs with the class 0 to 9 respectively as the last value of the array. The classified SDR data is given below.

```
data[0] = { 7, 18, 24, 29,......, 1012, 0 };
data[1] = { 25, 31, 44, 48,...... ,188, 1 };
data[2] = { 118, 123, 127, 156,........., 340, 2 };
data[3] = { 240, 242, 257, 266,.........., 444, 3 };
data[4] = { 302, 314, 324, 327,......., 518, 4 };
data[5] = { 393, 405, 428, 429,........, 624, 5 };
data[6] = { 483, 487, 500, 509,........., 726, 6 };
data[7] = { 579, 587, 595, 607,............, 814, 7 };
data[8] = { 676, 691, 700, 707,......... ,  916, 8 };
data[9] = { 772, 779, 780, 800,............., 1007, 9 };
```

Table 2: Text File & Old Dataset

We train the model based on these classified SDR's, then evaluate the model using the unclassified data as given below.

Unclassified = {461, 495, 515, 501,…, 712 }

The model successfully classified the unclassified data using KNN classifier and labeled the unclassified data with class 6 as the data point of unclassified data resembles closely with the SDR's having class label as class 6 in classified training data. The value of K is 1 in this case, The reason for using is that the training dataset is so limited, every data in the training dataset has a separate class so the algorithm should only consider the first nearest neighbor, as classify the unclassified dataset.

With this approach, we can classify the value of the sequence as instructed, but we want to solve some real-life problems out of this, so after some research, we came to an idea that we want to be executed with this approach but in a much better way. The idea was to predict or identify the whole sequence that belongs to sets of numbers. This approach is being discussed in the next section.

2. Classification of different Sets of Number sequences based on Sparse Distributed Representations (SDRs)

To implement this approach, we generate SDR from different sequences S1, S2, and S3, S1 is of even number, S2 is an odd number, and S3 is a decimal number that is Neither odd nor even. The SDRs are extracted with a multi-sequence learning experiment. At first, we just take one SDR for each sequence and use it to an unclassified sequence that contains most of the elements from one of the sets, with this we can classify the unclassified sequence. To make the model more efficient we must increase the dataset. To increase the dataset, we generate SDR 11 times of the same sequences. 11 time is not fixed but we just increase the dataset that many generations. Then we collect this data first in a text file. We validate the model by splitting the dataset in a 70- 30 splitting ratio. 70% of the dataset in the text file is used to train the model while 30% remains to evaluate the model performance and prediction. The algorithm randomly splits the data to evaluate the model performance unbiasedly, each testing data is classified based on the train training dataset. The model predicts the class label of every test data almost every time accurately.

To improve the structure of data we came to the idea that it's better to have the dataset in a JSON file rather than a text file. In the text file we label the SDRs with 0, 1, and 2, 0 is considered to be S1 or even, 1 is considered to be S2 or odd and 2 is considered to be Neither odd nor even which we evaluate when we are generating the SDR's but for us, it was not considered to be a great approach so we change the file type to JSON which help us to read data in more structure and reliable way. The sample dataset in the JSON file we created is shown in Table 1.

Using this we can map SDRs to sequences exactly. To consider the optimal value of K, we cross-validate the value of K for different test datasets which is defined and a cross-validation table is described in the result section.

C. KNN Code References

At the high level, If I summarize the pseudo-code, there are only three major steps which include:

```
1. Compute distances from unknown
2. Sort the distances (nearest, farthest)
3. Use a vote to determine the result
```

These sections try to discuss the methods defined within the KNN classifier Class; these methods are discussed below:

1) Distance(double[] vector1, double[] vector2): As we have discussed in section 2(A) regarding the Euclidean distance. The distance function calculates the Euclidean distance between two vectors which in our case are vector1 and vector2. [7] As the formula of Euclidean distance is to go through each element of a vector compute the squared difference and then find the accumulative sum. And then finally finding the square root of the whole sum, represents the Euclidean distance between two vectors.

```
FUNCTION CalculateEuclideanDistance(testFeature, trainFeature):
IF testFeature IS NULL OR trainFeature IS NULL
THEN
THROW ArgumentNullException
END IF

IF testFeature.Count <> trainFeature.Count
THEN
THROW ArgumentException
END IF

SET sumOfSquaredDifferences TO 0.0

FOR EACH index FROM 0 TO testFeature.Count - 1
DO
        SET difference TO testFeature[index] trainFeature[index]
        SET squaredDifference TO difference * difference
        ADD squaredDifference TO
sumOfSquaredDifferences END FOR

SET euclideanDistance TO SquareRoot( sumOfSquaredDifferences)

        RETURN euclideanDistance END
FUNCTION
```

This is a simple implementation of distance finding by Euclidean. We can also use other alternatives including Mahalanobis distance, Mahattan distance. K-NN is not good for mixed numerical and non-numerical data (categorical data), as K-NN needs the two notions"nearest" and most distance metrics to work so it strictly works on either numerical data or non-numeric data.

2) Vote(IndexAndDistance[] info, double[][] trainData, int numClasses, int k): This method performs the voting mechanism between the k nearest neighbors. The arguments of this method are IndexAndDistance which represents distance and Indices of nearest neighbors in other we can say the information of nearest neighbors, then our training dataset (training-features), the total number of classes (numClasses), and the value of K.

The step of the Vote method is to initialize an array whose responsibility is to store the total number of votes against each class means every nearest k-neighbor has a vote. In the starting the array is initialized as zero. Then in the second step, it goes through the first k neighbors and retrieves its class label from the training data, and the respective array index is counted as one positive vote. In our case, the training data have either an odd number or an even number respective voting has been counted for each of them. Then the counted voting with the highest vote returns its label as the predicted class.

```
FUNCTION Vote(nearestNeighbors, trainingLabels, k) :
CREATE votes Dictionary
FOR EACH label IN trainingLabels DO votes[label] = 0
END FOR

FOR i FROM 0 TO k - 1 DO
        SET neighborLabel TO trainingLabels[
nearestNeighbors[i].Index]
INCREMENT votes[neighborLabel]
END FOR
SET classWithMostVotes TO label WITH MAXIMUM votes

OUTPUT "Predicted class for test data:"
IF classWithMostVotes == "S1" THEN
OUTPUT "S1 = {2, 4, 6, 8, 10, 12, 14} (
Even) Sequence Match"
ELSE IF classWithMostVotes == "S2" THEN
OUTPUT "S2 = {3, 5, 7, 9, 11, 13, 15} (Odd) Sequence Match"
ELSE IF classWithMostVotes == "S3" THEN
OUTPUT "S3 = { 4.5, 11.4, 12.8, 15.5, 16.6, 17.7 } (Neither Odd nor
Even) Sequence Match"
ELSE
OUTPUT "MisMatch"
END IF

RETURN INDEX of classWithMostVotes in trainingLabels

END FUNCTION
```

As we know finding a class label is a little bit trickier from the k items. Every k-nearest training receives one vote for its class label, as demonstrated by the code above. At the start, votes are initialized as zero. Then it adds if the class is nearest. As currently we also have to count the votes. So, the class that has a higher number of votes is the class from where our test data belongs from that specific class. It can be either odd, even, or decimal class.

3) *Classifier(double[] unknownSDR, double[][] Sdrdata, int numofclass, int k):*

This method performs the KNN classification for a given unknown SDR which we are also saying as test-data. The arguments of this method include the testdata (unknownSDR) and the training data (Sdrdata) and the total number of classes which is 3 in our case. And the value of k as an input parameter. In this method, we have called all the methods including IndexandDistance objects to store indices and distances of all data points from the unknown SDR.

The next, we have called the distance method, which will calculate the distances between the test data ad training data points that are stored in the array. Then we sorted an array in ascending order based on distances.

Then in the final, we have called the vote method to determine the predicted class label based on k nearest neighbors. Then, the class with the highest vote is where the test data belongs to that specific class. And then this method returns the result of a specific class label.

4) *IndexAndDistance Class*

This class is used to represent the index of a training item and its distance from the input (test data). It implements the ComparableIndexAndDistance Interface to enable sorting based on distance. The CompareTo method is overridden to compare distances between instances of the IndexAndDistance class.

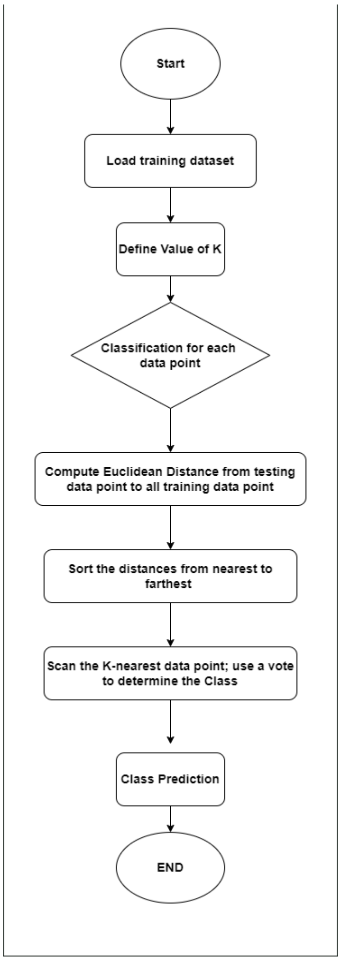
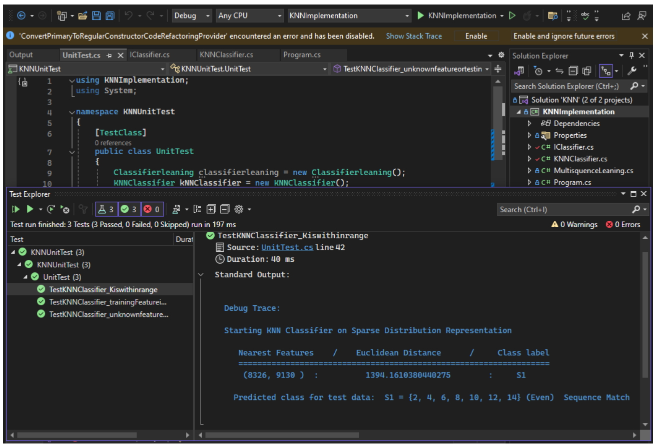

Start
Load training dataset
Define Value of K
Classification for each data point
Compute Euclidean Distance from testing data point to all training data point
Sort the distances from nearest to farthest
Scan the K-nearest data point; use a vote to determine the Class
Class Prediction
END

Figure 6: KNN Flow Diagram according to steps.

5) Unit Testing and Exception Handling

The outcomes of the unit test model are depicted in Figure 10. Unit test is developed to handle various scenarios, including handling exceptions. All the unit tests that are designed and implemented for the HTM classifier have been integrated into the KNN classifier. During conducting the test, we did not notice any instances of failed tests. Indicating the robustness and reliability of the implemented algorithm. But there is one important thing is that the continuous testing and refinement of the model are imperative to ensure its accuracy and performance across diverse scenarios and datasets.



'ConvertPrimaryToRegularConstructorCodeRefactoringProvider' encountered an error and has been disabled.

```
using KNNImplementation;
using System;

namespace KNNUnitTest
{
    [TestClass]
    0 references
    public class UnitTest
    {
        Classifierleaning classifierleaning = new Classifierleaning();
        KNNClassifier kNNClassifier = new KNNClassifier();
```

Test run finished: 3 Tests (3 Passed, 0 Failed, 0 Skipped) run in 197 ms

```
TestKNNClassifier_Kiswithinrange
Source: UnitTest.cs line 42
Duration: 40 ms

Standard Output:


  Debug Trace:

  Starting KNN Classifier on Sparse Distribution Representation

     Nearest Features    /    Euclidean Distance      /     Class label
     ================================================================
      (8326, 9130 )  :          1394.1610380440275        :     S1

    Predicted class for test data:  S1 = {2, 4, 6, 8, 10, 12, 14} (Even)  Sequence Match
```

Figure 7: Unit Testing and Exception Handling

IV. Results

In our scenario, we utilize a data splitting method that randomly shuffles the dataset's rows, allocating a specified ratio for training and the remainder for testing. Specifically, 70% of the data is allocated for training, while 30% is reserved for testing. The method returns the training and testing datasets as separate Lists. The reported accuracy values represent the model's prediction accuracy at various values of k when tested with randomly generated test data. These accuracy percentages provide insights into the performance of our model under different configurations of the k-Nearest Neighbors algorithm.

K	Accuracy	Random Generated Test Data Accuracy in Percentage						
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
1	100	90.9	90.9	90.9	90.9	90.9	90.9	100
2	100	100	90.9	90.9	90.9	90.9	90.9	100
3	100	100	100	100	100	90.9	90.9	100
4	90.9	100	100	90.9	90.9	90.9	90.9	100
5	100	100	100	90.9	100	90.9	90.9	100

Table 3: Accuracy of the KNN Classifier for Different Values of K for Different Testing Data

The analysis presented in Table 1 suggests that the optimal value of k for the k-Nearest Neighbors (k-NN) classifier, when applied to a dataset with three classes, is k=3. Here's a breakdown of the findings:

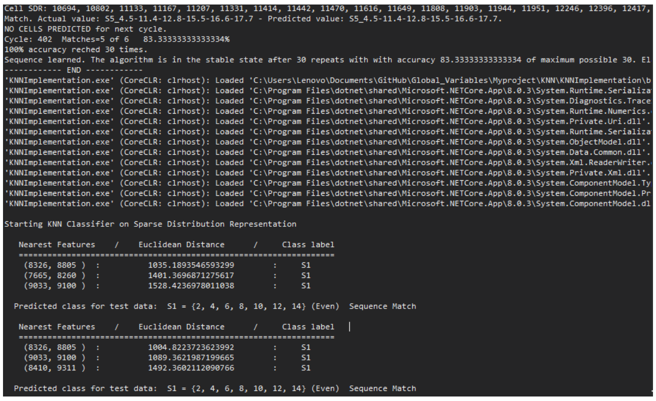
1) K=1, 2, and 4: When the model relies solely on the single nearest neighbor (k=1) or a small number of nearest neighbors (k=2, 4), the accuracy tends to hover around 90.9%.

This indicates that the classifier may be prone to misclassifications or inconsistencies when considering a small number of neighbors, leading to accuracy fluctuations.

2) K=3: At k=3, the accuracy consistently remains around 100% for most randomly generated testing data splits from the dataset. This suggests that considering the three nearest neighbors leads to more stable and reliable predictions, resulting in higher accuracy across various testing scenarios.

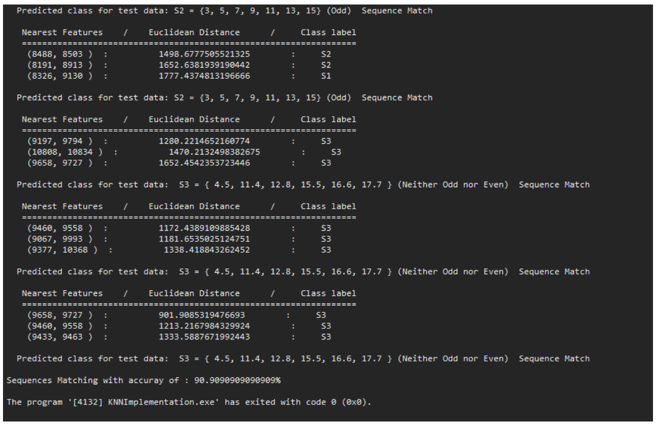
3) K=5: While the accuracy at k=5 is also reported as 100%, it is noted that there are instances where the accuracy drops to 90.9%. This indicates some variability in performance compared to k=3, where the accuracy remains consistently high.

Based on these observations, it's reasonable to conclude that k=3 appears to be the optimal choice for the k-NN classifier with this dataset.



```
Cell SDR: 10694, 10802, 11133, 11167, 11207, 11331, 11414, 11442, 11470, 11616, 11649, 11808, 11903, 11944, 11951, 12246, 12396, 12417,
Match. Actual value: S5_4.5-11.4-12.8-15.5-16.6-17.7 - Predicted value: S5_4.5-11.4-12.8-15.5-16.6-17.7.
NO CELLS PREDICTED for next cycle.
Cycle: 402  Matches=5 of 6   83.33333333333334%
100% accuracy reched 30 times.
Sequence learned. The algorithm is in the stable state after 30 repeats with with accuracy 83.33333333333334 of maximum possible 30. El
------------ END ------------
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Users\Lenovo\Documents\GitHub\Global_Variables\Myproject\KNN\KNNImplementation\b
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.Runtime.Serializa
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.Diagnostics.Trace
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.Runtime.Numerics.
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.Private.Uri.dll'.
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.Runtime.Serializa
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.ObjectModel.dll'.
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.Data.Common.dll'.
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.Xml.ReaderWriter.
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.Private.Xml.dll'.
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.ComponentModel.Ty
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.ComponentModel.Pr
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.3\System.ComponentModel.dl

Starting KNN Classifier on Sparse Distribution Representation

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (8326, 8805 )  :         1035.1893546593299       :     S1
    (7665, 8260 )  :         1401.3696871275617       :     S1
    (9033, 9100 )  :         1528.4236978011038       :     S1

  Predicted class for test data:  S1 = {2, 4, 6, 8, 10, 12, 14} (Even)  Sequence Match

   Nearest Features    /    Euclidean Distance      /     Class label   |
   ================================================================
    (8326, 8805 )  :         1004.8223723623992       :     S1
    (9033, 9100 )  :         1089.3621987199665       :     S1
    (8410, 9311 )  :         1492.3602112090766       :     S1

  Predicted class for test data:  S1 = {2, 4, 6, 8, 10, 12, 14} (Even)  Sequence Match
```

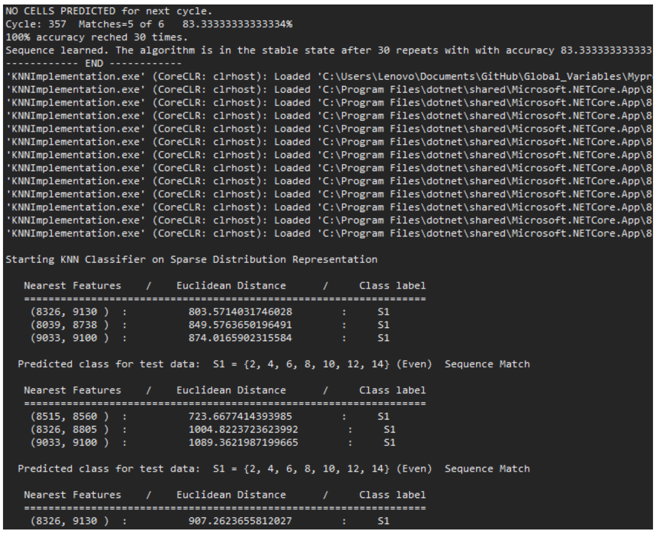
Figure 8: Predicted Result displayed on output Window (A)



```
  Predicted class for test data: S2 = {3, 5, 7, 9, 11, 13, 15} (Odd)  Sequence Match

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (8488, 8503 )  :         1498.6777505521325       :     S2
    (8191, 8913 )  :         1652.6381939190442       :     S2
    (8326, 9130 )  :         1777.4374813196666       :     S1

  Predicted class for test data: S2 = {3, 5, 7, 9, 11, 13, 15} (Odd)  Sequence Match

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (9197, 9794 )  :         1280.2214652160774       :     S3
    (10808, 10834 )  :         1470.2132498382675       :     S3
    (9658, 9727 )  :         1652.4542353723446       :     S3

  Predicted class for test data:  S3 = { 4.5, 11.4, 12.8, 15.5, 16.6, 17.7 } (Neither Odd nor Even)  Sequence Match

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (9460, 9558 )  :         1172.4389109885428       :     S3
    (9067, 9993 )  :         1181.6535025124751       :     S3
    (9377, 10368 )  :         1338.418843262452       :     S3

  Predicted class for test data:  S3 = { 4.5, 11.4, 12.8, 15.5, 16.6, 17.7 } (Neither Odd nor Even)  Sequence Match

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (9658, 9727 )  :         901.9085319476693       :     S3
    (9460, 9558 )  :         1213.2167984329924       :     S3
    (9433, 9463 )  :         1333.5887671992443       :     S3

  Predicted class for test data:  S3 = { 4.5, 11.4, 12.8, 15.5, 16.6, 17.7 } (Neither Odd nor Even)  Sequence Match

Sequences Matching with accuray of : 90.9090909090909%

The program '[4132] KNNImplementation.exe' has exited with code 0 (0x0).
```

Figure 9: Predicted Result displayed on output Window (B)

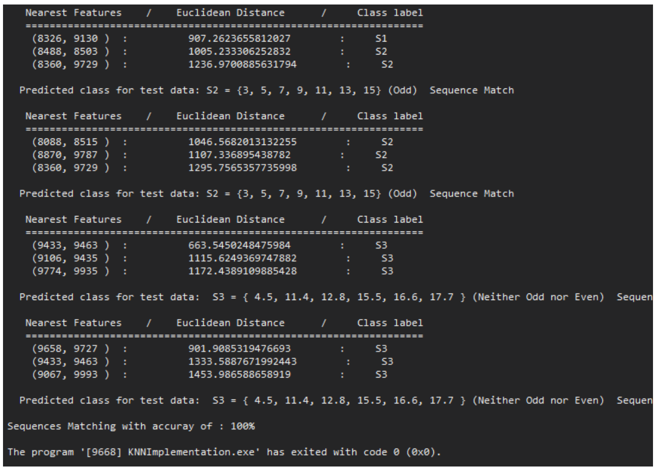
Figures 8 & 9 display the output window presenting the predicted results. They showcase the nearest neighbors for the test data point with a value of k set to 3, along with the calculated distances and the classification of the class for that specific test data. Subsequently, the predicted class for the test data is indicated, determining whether it is even, odd, or neither based on the voting method. This process repeats for each test data point. Finally, the model's predicted accuracy is displayed at the end.



```
NO CELLS PREDICTED for next cycle.
Cycle: 357  Matches=5 of 6   83.33333333333334%
100% accuracy reched 30 times.
Sequence learned. The algorithm is in the stable state after 30 repeats with with accuracy 83.333333333333
------------ END ------------
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Users\Lenovo\Documents\GitHub\Global_Variables\Mypr
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8
'KNNImplementation.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8

Starting KNN Classifier on Sparse Distribution Representation

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (8326, 9130 )  :         803.5714031746028       :     S1
    (8039, 8738 )  :         849.5763650196491       :     S1
    (9033, 9100 )  :         874.0165902315584       :     S1

  Predicted class for test data:  S1 = {2, 4, 6, 8, 10, 12, 14} (Even)  Sequence Match

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (8515, 8560 )  :         723.6677414393985       :     S1
    (8326, 8805 )  :         1004.8223723623992       :     S1
    (9033, 9100 )  :         1089.3621987199665       :     S1

  Predicted class for test data:  S1 = {2, 4, 6, 8, 10, 12, 14} (Even)  Sequence Match

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (8326, 9130 )  :         907.2623655812027       :     S1
```

Figure 10: Predicted Result displayed on output Window (A)

Figure 7, depicts the unit test conducted on the KNN classifier. A random SDR is selected from the dataset, serving as the test data list to evaluate the classifier's performance. The test ensures that the predicted value by the classifier aligns with the actual class value of the test data. The KNN classifier successfully passes the unit test, as evident from Figure 7. Additionally, we've incorporated an exception in the unit test to accommodate varying values of K. If the value of K surpasses the length of the SDR data, the test gracefully handles this scenario.



```
   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (8326, 9130 )  :         907.2623655812027       :     S1
    (8488, 8503 )  :         1005.233306252832       :     S2
    (8360, 9729 )  :         1236.9700885631794       :     S2

  Predicted class for test data: S2 = {3, 5, 7, 9, 11, 13, 15} (Odd)  Sequence Match

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (8088, 8515 )  :         1046.5682013132255       :     S2
    (8870, 9787 )  :         1107.336895438782       :     S2
    (8360, 9729 )  :         1295.7565357735998       :     S2

  Predicted class for test data: S2 = {3, 5, 7, 9, 11, 13, 15} (Odd)  Sequence Match

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (9433, 9463 )  :         663.5450248475984       :     S3
    (9106, 9435 )  :         1115.6249369747882       :     S3
    (9774, 9935 )  :         1172.4389109885428       :     S3

  Predicted class for test data:  S3 = { 4.5, 11.4, 12.8, 15.5, 16.6, 17.7 } (Neither Odd nor Even)  Sequen

   Nearest Features    /    Euclidean Distance      /     Class label
   ================================================================
    (9658, 9727 )  :         901.9085319476693       :     S3
    (9433, 9463 )  :         1333.5887671992443       :     S3
    (9067, 9993 )  :         1453.986588658919       :     S3

  Predicted class for test data:  S3 = { 4.5, 11.4, 12.8, 15.5, 16.6, 17.7 } (Neither Odd nor Even)  Sequen

Sequences Matching with accuray of : 100%

The program '[9668] KNNImplementation.exe' has exited with code 0 (0x0).
```

Figure 11: Predicted Result displayed on output Window (B)

V. APPLICATION OF KNN

KNN Algorithm is utilized in different applications across different sectors, mostly in classification. [20]The common cases include:

A. Pattern Recognition

KNN is used for the identification of specific patterns, it can be in a text. Like it predicts the missing fill-in-the-blanks. This is help in solving cache which are handwritten numbers. So, KNN can also identify these patterns.

B. Healthcare

The common use of KNN is the health department's prediction of chances of cancer as well as heart attack risks.[53] The algorithm tries to learn from the most likely gene expressions.

C. Recommendation Engines

When we surf on internet, the KNN algorithm can be used by the website to recommend other additional content. This recommendation is based on user behavior. However, for larger datasets, this approach is not optimal.

D. Data Preprocessing

Mostly we have missing values in our dataset, KNN algorithm can help to determine those values. Those estimated missing values are also known as missing data imputation.

E. Finance

The banks use credit card spending to predict the risk associated with the specific individual loan. As well as KNN is used to determine the credit card worthiness of a loan application. So, KNN is used in a variety of economic and finance departments. As well as the other common use cases are currency exchange forecasting, stock market as well as trading futures, etc. [20]

VI. Conclusion

Firstly, we designed a straightforward KNN prototype algorithm aimed at predicting sequences. One SDR array for each sequence, we tested the model against slightly mismatched sequences, achieving the desired outcomes. To enhance the model's robustness, we compiled a comprehensive dataset consisting of SDR values corresponding to various types of sequences: even numbers, odd numbers, and decimals. This dataset enabled the model to effectively address the classification challenges posed by these sequence categories. Consequently, the model exhibits remarkable predictive accuracy. While the model consistently achieves near-perfect predictions, occasionally reaching 100% accuracy, there were rare instances where predictions hovered around 90.9%, still maintaining the highest level of accuracy attainable. Furthermore, we implemented unit tests to handle special cases, drawing upon the HTM Classifier for reference. These tests have yielded satisfactory results, further bolstering the reliability and performance of our model.

VII. References

[1] N. F. F. E. B. S. L. P. d. L. Marcelo Beckmann, "A KNN Undersampling Approach for Data Balancing," *Journal of Intelligent Learning Systems and Applications,* vol. Vol.7 No.4, November 11, 2015.

[2] T. &. H. P. Cover, "Nearest neighbor pattern classification.," *IEEE transactions on information theory, 13(1), 21-27.,* 1967.

[3] K. G. K. V. Han EH.., ""Text Categorization Using Weight Adjusted k-Nearest Neighbour Classification".," *In: Cheung D., Williams G.J., Li Q. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2001. Lecture Notes in Computer Sci,* 2001.

[4] Z. Z., " Introduction to machine learning: k-nearest neighbors.," *Ann Transl Med. 2016 Jun;4(11):218.,* no. doi: 10.21037/atm.2016.03.37. PMID: 27386492; PMCID: PMC4916348..

[5] J. H. D. Z. H. Z. a. C. L. G. Song Yang, " "KNN: Informative k-nearest neighbour pattern classification,"," *Knowledge Discovery in Databases, ,* p. Pg. 248– 264., 2007.

[6] N. A. M. G. R. M. J. Norsyela Muhammad Noor Mathivanan, "A comparative study on dimensionality reduction between principal component analysis and k-means clustering," *Indonesian Journal of Electrical Engineering and Computer Science 16(2):752 ,* Vols. 10.11591/ijeecs.v16.i2.pp752-758, November 2019.

[7] "Euclidean Distance, https://byjus.com/maths/euclidean-distance/".

[8] L. &. L. C. &. M. N. &. M. A. Liberti, "Euclidean Distance Geometry and Applications," vol. SIAM Review. 56. 10.1137/120875909., 2012.

[9] R. &. S. Z. &. Z. E. Suwanda, "Analysis of Euclidean Distance and Manhattan Distance in the K-Means Algorithm for Variations Number of Centroid K. Journal of Physics: Conference Series. 1566. 012058. 10.1088/1742-6596/1566/," 2020.

[10] "How to Decide the Perfect Distance Metric For Your Machine Learning Model,," *https://www.turing.com/kb/how-to-decide-perfect-distance- metric-for-machine-learning-model.*

[11] B. &. C. M. &. B. C. &. H. P. Lu, "The Minkowski approach for choosing the distance metric in geographically weighted regression.," *International Journal of Geographical Information Science. ,* Vols. 30. 1-18. 10.1080/13658816.20, 2015.

[12] A. &. K. V. &. R. T. Bookstein, "Generalized Hamming Distance.," vol. Information Retrieval. 5. 10.1023/A:1020499411651., 2002.

[13] K. Bala Priya C, "Distance Metrics: Euclidean, Manhattan, Minkowski,," *https://www.kdnuggets.com/2023/03/distance-metrics-euclidean-manhattan-minkowski-oh.html.*

[14] "What is the k-nearest neighbors (KNN) algorithm?," *https://www.ibm.com/topics/knn.*

[15] J. R. S. H. G. S. R. Goldberger, "Neighbourhood components analysis.," *NIPS (2004).*

[16] Y. J. D. &. C. D. Cai, "A KNN Research Paper Classification Method Based on Shared Nearest Neighbor.," *NTCIR Conference on Evaluation of Information Access Technologies.,* 2010.

[17] D. Dobric, "Influence of input sparsity to Hierarchical Temporal Memory Spatial Pooler Noise Robustness," 2019.

[18] J. H. a. D. George, ""Hierarchical Temporal Memory Concepts, Theory and Terminology"," *Numenta, Redwood City, CA, USA,* no. Available: http://www.mlanctot.info/files/papers/NumentaHTMConcepts.pdf, 2006.

[19] S. A. a. J. Hawkins, ""Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory"," vol. http://arxiv.org/abs/1503.07469, 2015.

[20] S. &. B. M. Imandoust, "Application of K-nearest neighbor (KNN) approach for predicting economic events theoretical background.," *Int J Eng Res Appl. 3. 605-610.,* 2013.

[21] J. Z. A. L. J. W. Q. a. L. S. Zhang, "An adaptive KNN algorithm based on data point density," *IEEE Access, 7, 80876-80883. doi: 10.1109/AC- CESS.2019.2926611,* 2019.

[22] S. A. a. J. H. Y. Cui, ""Continuous Online Sequence Learning with an Unsupervised Neural Network Model,"," *Neural Comut.,, vol. 28, pp. 2474- 2504.,* 2016.

[23] T. T. R. a. F. J. Hastie, "The elements of statistical learning: Data mining, inference, and prediction.," 2009.

[24] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression.," *The American Statistician, 46(3), 175-185.,* 1992.

[25] G. W. D. H. T. a. T. R. James, "An introduction to statistical learning.," 2013.

[26] X. L. Y. a. C. Z. Yu, ""Weighted K-nearest neighbor algorithm based on Gaussian kernel function"," *Journal of Ambient Intelligence and Hu- manized Computing, 11(5), 1985-1993. doi: 10.1007/s12652-019-01633,* 2020.

[27] K. a. D. S. a. V. S. a. S. A. Taunk, "A Brief Review of Nearest Neighbor Algorithm for Learning and Classification," Vols. 1255-1260. 10.1109/ICCS45141.2019.9065747., 2019.

[28] D. W. a. D. T. G., "“Locally adaptive nearest neighbour algorithms”," *Adv. Neural Inf. Process. Syst,* p. pg. 184–186, 1994.

[29] C. M. Bishop, "Pattern recognition and machine learning.," 2006.

[30] Y. a. K. J. T. Zhang, "Content-based image retrieval using edge-based color histogram and adaptive similarity measure.," *In Proceedings of the eleventh ACM international conference on Multimedia.,* 2003.

[31] H. Y. L. a. L. H. Liu, "Discretization: An enabling technique.," *Data Mining and Knowledge Discovery, 6(4),* pp. 393-423., 2005.

[32] J. Brownlee, "Machine learning mastery with Python: Understand your data, create accurate models and work projects end-to-end.," 2016.

[33] A. C. a. G. S. Mueller, "Introduction to machine learning with Python: A guide for data scientists.," 2016.

[34] W. a. o. McKinney, "Python for data analysis: Data wrangling with Pandas, NumPy, and IPython.," 2017.

[35] F. V. G. G. A. M. V. T. B. G. O. .. a. D. E. Pedregosa, "Scikit-learn: Machine learning in Python.," *Journal of Machine Learning Research,* Vols. 12(Oct), 2825-2830., 2011.

[36] A. a. o. Kulkarni, "Building intelligent systems: A guide to machine learning engineering.," 2018.

[37] J. a. D. W. &. S. N. Sun, "A Survey of kNN Algorithm.," *Information Engineering and Applied Computing. ,* vol. 1. 10.18063/ieac.v1i1.770., 2018.

[38] S. D. S. V. a. A. S. K. Taunk, "”A Brief Re- view of Nearest Neighbor Algorithm for Learning and Classification,” Madurai, India,," *International Conference on Intelligent Computing and Control Systems (ICCS), ,* pp. pp. 1255-1260, , 2019 .

[39] P. S. P. K. D. Komal Bharti, "”Comparison of Multiple Classifiers for Audio-Visual Speaker Recognition System”," *International Conference on Computational Intelligence, Networks and Security (ICCINS),* pp. pp.1-6, 2023.

[40] E. Ozturk Kiyak, B. Ghasemkhani and Birant, " High-Level K-Nearest Neighbors (HLKNN): A Supervised Machine Learning Model for Classification Analysis.," *Electronics,* no. https://doi.org/10.3390/electronics12183828, p. 3828, 12, 2023.

[41] T. H. P. Cover, "Nearest neighbor pattern classification.," *IEEE Transactions on Information Theory 13(1),* pp. 21-27, 1967.

[42] Y. Z. S. Z. X. Z. J. Z. C. Qin, "Semi-parametric optimization for missing data imputation.," *Applied Intelligence 27,* p. 79–88 , 2007.

[43] K. B. J. S. L. Weinberger, "Distance metric learning for large margin nearest neighbor classification.," *NIPS,* p. pp. 1473–1480, 2005.

[44] X. Z. C. Z. S. Wu, "Efficient mining of both positive and negative association rules.," *ACM Transactions on Information Systems (TOIS) 22(3),* p. 381–405, 2004.

[45] C. Z. X. Z. J. Q. Y. Z. S. Zhang, "An imputation method for missing values.," *Zhou, Z.-H., Li, H., Yang, Q. (eds.) PAKDD 2007. LNCS (LNAI),* Vols. vol. 4426, . Springer, , p. pp. 1080–1087, Heidelberg (2007).

[46] S. Zhang, "KNN-CF approach: Incorporating certainty factor to knn classification.," *IEEE Intelligent Informatics Bulletin 11(1),* p. 24–33, 2010.

[47] S. Zhang, "Nearest neighbor selection for iteratively knn imputation.," *Journal of Systems and Software,* p. 2541–2552 , 2012.

[48] X. Z. L. H. Z. Zhu, "A sparse embedding and least variance encoding approach to hashing.," *IEEE Transactions on Image Processing 23(9),* p. 3737–3750 , 2014.

[49] G. K. V. K. E.-H. (. Han, "Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification," *Department of Computer Science and Engineering University of Minnesota, USA, ,* Minnesota, 1999..

[50] J. H. a. D. George, "Hierarchical Temporal Memory Concepts, Theory and Terminology," *Numenta, Redwood City, CA, USA,* no. http://www.mlanctot.info/files/papers/NumentaHTMConcepts.pdf, 2006.

[51] A. Rodin, "Unsupervised real time anomaly detection," *https://www.griddynamics.com/blog/unsupervised-real-time-anomaly-detection.*