Investigate and Implement KNN Classifier

1st Kejdjon Cela
kejdjon.cela@stud.fra-uas.de

2nd Mariglen Xhelo
mariglen.xhelo@stud.fra-uas.de

***Abstract*—Machine learning (ML) has seen widespread adoption across various sectors, including finance, education, communication, transportation, retail, and healthcare. Researchers have developed numerous algorithms to analyze extensive datasets and extract actionable insights. ML encompasses algorithms tailored for text, image, audio, video, and numeric data formats, with applications ranging from prediction to classification. Supervised learning, a prominent ML type, employs techniques like classification and regression for data analysis. Notable classifiers in supervised machine learning include decision trees, support vector machines, naive Bayes, and K-nearest neighbors (KNN). This paper proposes implementing the widely used KNN algorithm to predict outcomes based on input variables. The KNN classifier can seamlessly integrate with HTM, serving as a sub-module for classification tasks. Specifically, HTM models learn to represent input data in a high-dimensional feature space, and the KNN classifier then leverages this representation to classify data based on nearest neighbors. This integration proves beneficial for datasets with complex temporal structures, requiring advanced classification approaches beyond simplistic threshold-based methods. The developed KNN model is integrated into the Neocortex API to receive input datasets from Hierarchical Temporal Memory (HTM). The model successfully processes sequences from HTM and classifies them as matching or mismatching with the input data sequences.**

Keywords—Hierarchical Temporal Memory (HTM), Neocortex API, Machine Learning, Supervised Learning, K-Nearest Neighbors, Classification, Prediction.

I. Introduction

Machine learning algorithms typically operate on datasets where each instance is characterized by a consistent set of features, which can be continuous, categorical, or binary. When these instances are accompanied by known labels, the learning process is termed supervised [1]. Conversely, unsupervised learning occurs when instances lack labels. Through unsupervised algorithms like clustering, researchers aim to identify previously unknown yet valuable item classes. Reinforcement learning represents another form of machine learning, where the training input, provided by the environment, comes in the form of a scalar reinforcement signal, indicating the system's performance. Unlike supervised learning, the learner isn't directed on which actions to take but must deduce the best actions through trial and error to maximize rewards. Intelligent systems often prioritize supervised classification as one of their primary tasks [2]. Statistical approaches in classification are distinguished by their reliance on explicit underlying probability models. These models offer probabilities indicating the likelihood of an instance belonging to each class, rather than just a straightforward classification. Within this classification framework, examples include Bayesian networks and instance-based methods. Instance-based learning algorithms are often referred to as lazy-learning algorithms because they postpone the induction or generalization process until classification is required. Unlike eager-learning algorithms like decision trees, neural networks, and Bayes nets, lazy-learning algorithms demand less computational time during the training phase but typically require more computational time during classification [3]. One of the simplest instance-based learning algorithms is the nearest neighbor algorithm.

Additionally, in exploring various machine learning methodologies, it's essential to consider Hierarchical Temporal Memory with Cortical Learning Algorithm (HTM-CLA). HTM-CLA is an innovative approach inspired by the structure and function of the human neocortex. It offers a promising foundation for developing machines capable of achieving or surpassing human-level performance across a range of cognitive tasks. HTM-CLA is implemented within the NuPIC, an open-source project spearheaded by Numenta [4].

A. Machine Learning Classification

Classification is a supervised machine learning method where the model tries to predict the correct label of a given input data. In classification, the model is fully trained using the training data, and then it is evaluated on test data before being used to perform prediction on new unseen data. For instance, an algorithm can learn to predict whether a given email is spam or ham (no spam).

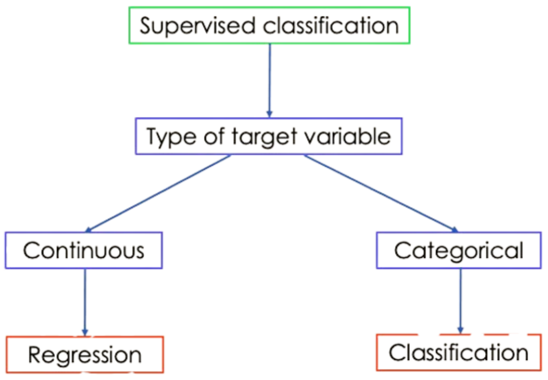

Fig. 1. Supervised learning classification.

Machine learning algorithms are broadly categorized into four main types: supervised, unsupervised, semi-supervised, and reinforcement learning. While both classification and regression fall under supervised learning see “Fig. 1”, they serve different purposes. Classification involves predicting discrete target variables, such as determining the sentiment of a text. On the other hand, regression focuses on predicting continuous target variables, like forecasting a person's salary based on factors such as education level, work experience, location, and seniority level.

B. *Hierarchical Temporal Memory (HTM)*

Th The recent advancements in the theory of Hierarchical Temporal Memory (HTM) - Cortical Learning Algorithms (CLA) have introduced a revolutionary paradigm in the field of machine intelligence. HTM-CLA, inspired by the memory-prediction theory of brain function proposed by Jeff Hawkins, is a biomimetic model aimed at discovering and inferring the underlying causes of observed input patterns and sequences, thereby constructing a progressively intricate model of the world. Essentially, HTM-CLA offers a theoretical framework for understanding the workings of the human brain. There are three fundamental characteristics of the brain that play a crucial role in the design of HTM-CLA. Firstly, the brain operates as a hierarchical system, facilitating signal flow in both upward and downward directions within the hierarchy, as well as within regions themselves [5]. Secondly, temporal information is inherent in all aspects of brain learning, emphasizing the significance of time in processing information. Thirdly, the human brain fundamentally functions as a memory system, constantly endeavoring to remember and predict patterns over time. The cells and connections within the brain store these patterns experienced over time. Utilizing a hierarchical structure offers numerous advantages for learning.

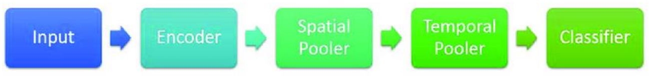

Fig. 2. NuPIC pipeline process flow.

The NuPIC framework incorporates the implementation of HTM-CLA, facilitating the modeling of structures and sequences based on memory predictions. In this framework, the process flow involves several key components as in shown “Fig. 2”:

- **Encoders**: Input data are initially processed by encoders, which convert them into binary bits based on the semantics of the variable. Similar values result in overlapping bits.
- **Spatial Pooler:** This component corresponds to the learning function of columns within the region. Its purpose is to establish connections between input bits and columns, aiming to form a stable representation of input patterns.
- **Temporal Pooler**: Responsible for enabling cells within a column to learn to represent input in the context of time.
- **Classifier**: The classifier attempts to infer the output from active columns in the upper region of the hierarchy.

However, while HTM-CLA holds promise in learning and representing complex patterns, the classifiers utilized for inferring classification output have shown limitations. The knnClassifier, for instance, maps nearest neighbors but necessitates storing every data point in memory, resulting in high complexity for large datasets and uncertain performance.

II. K-NEAREST-NEIGHBORS (KNN)

The K-Nearest-Neighbors (KNN) algorithm is a nonparametric classification method, meaning it doesn't make any assumptions about the underlying dataset distribution. It is praised for its simplicity and effectiveness in predicting the class of unlabeled data. KNN operates as a supervised learning algorithm, utilizing a labeled training dataset where data points are already categorized into different classes, enabling the prediction of the class for unlabeled data.

In classification tasks, various attributes are considered to determine the class assignment of unlabeled data. KNN is primarily employed as a classifier, tasked with categorizing data by leveraging the nearest or closest training examples within a defined region. This approach is favored for its straightforward implementation and minimal computational requirements. When dealing with continuous data, KNN employs the Euclidean distance metric to calculate the proximity of its nearest neighbors.

A. *WORKING*

The k-NN algorithm is primarily a classification method, which typically involves two main stages: the learning step and the assessment of the classifier. During the learning step, the algorithm constructs a classifier using the provided training data. In the case of the nearest neighbor technique, classification of new, unlabeled data is achieved by examining the classes of its nearest neighbors. The k-NN algorithm operates based on this principle, where a specific value of k is predetermined to aid in classifying unknown data points.

When encountering a new, unlabeled data point, the k-NN algorithm executes two key operations. Firstly, it identifies the k nearest neighbors to the new data point from the dataset. Secondly, it determines the class to which the new data should be assigned based on the classes of these neighboring points. Figure 1 illustrates a basic structure of the k-NN algorithm.

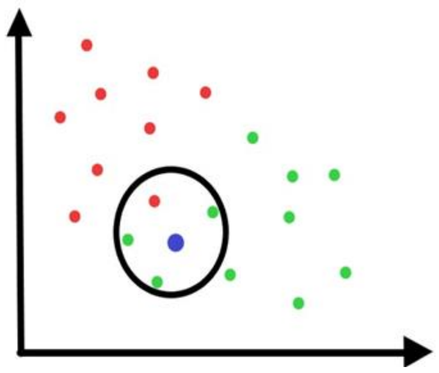

Fig. 3. A simple KNN.

When new data is introduced see “Fig. 3”, the k-NN algorithm classifies it based on its proximity to existing data points. It is particularly effective in datasets that exhibit clear clusters and are confined to specific regions within the data plot. Consequently, this algorithm enhances the accuracy of dividing data inputs into distinct classes with greater clarity. In k-NN classification, the algorithm determines the class that contains the highest number of points closest to the data point being classified. This involves calculating the Euclidean distance between the test sample and the specified training samples to ascertain their proximity.

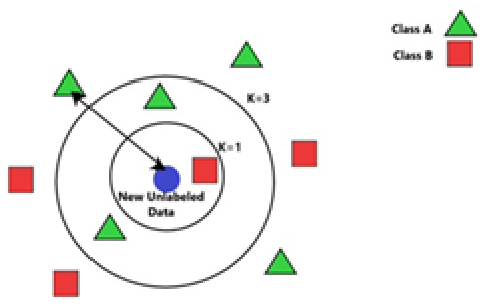

Fig. 4. KNN with two classes A and B.

As depicted in “Fig. 4” when dealing with new unlabeled data, the algorithm calculates its distance from each of its neighbors based on the specified value of K. Subsequently, it identifies the class that encompasses the maximum number of nearest neighbors. Once the K-nearest neighbors are gathered, the algorithm predicts the class of the training example by simply selecting the majority class among them. Several factors influence the performance of the KNN algorithm, including the choice of K, the calculation of Euclidean distance, and the normalization of parameters. To grasp the intricate workings of the algorithm, the following steps outline its detailed procedure [6].

Given the training dataset:

{(x(1), y(1)) , (x(2), y(2)), …… , (x(m), y(m))

}

Step1: Store the training set

Step2: For each new unlabeled data,

Calculate Euclidean distance with all training data points using the formula:

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad (1)$$

Find the k- nearest neighbours

When assigning a class to new data based on the maximum number of nearest neighbors, it's essential to normalize all parameters after storing the training data. This normalization simplifies calculations. The choice of the parameter 'K' significantly impacts the outcome of the classification. 'K' dictates the number of neighbors considered, thus influencing the boundaries of each class. Determining the optimal 'K' value involves examining the data and can be refined through cross-validation. When 'K' equals 1, data is allocated to the class of its nearest neighbor, resulting in a zero-error rate for the training data. However, this may lead to overfitting since boundaries become too tightly fitted to the training data. Conversely, very small 'K' values make the algorithm excessively sensitive to noise. Therefore, selecting an appropriate 'K' value requires careful consideration to balance precision and robustness in classification.

To determine an optimal value for 'K', it's crucial to partition the initial dataset into separate training and validation sets. When 'K' is set to 2 and the two nearest neighbors belong to different classes, the resulting outcome becomes uncertain. To address this ambiguity, increasing 'K' to a larger value, such as 5-nearest neighbors, can help define the nearest neighbor region more distinctly and provide clearer insights. However, larger values of 'K' may smooth out class boundaries excessively, potentially incorporating points from other classes into the neighborhood, which may not be desirable. Additionally, determining the appropriate value of 'K' becomes challenging when training data points are scattered throughout the dataset.

B. KNN AND ITS VARIANTS

1. **The locally adaptive kNN** algorithm, as introduced by [7], selects the appropriate value of 'k' for classifying an input by assessing cross-validation outcomes within the local neighborhood of the unlabeled data.

2. **Weight Adjusted KNN** algorithm proposed by [8] proposes transforming distances used in the search for nearest neighbors into weight measures. These weights determine the influence of each attribute on the classification process. This classifier is beneficial for datasets with numerous features, some of which may be deemed unnecessary, but it may come with a high computational cost.

3. **Improved KNN for Text** Categorization in [9], an enhanced KNN algorithm tailored for text categorization is introduced, combining KNN text categorization with a restricted one-pass clustering algorithm to construct the classification model. The traditional KNN approach, where a constant value of 'K' is applied to all classes, may favor classes with a higher number of attributes. However, in the improved KNN method, the appropriate number of nearest neighbors is selected based on the distribution of data within the training set to accurately predict the class of unlabeled data.

4. **Adaptive KNN**, as proposed by [10], departs from the traditional approach where the same number of nearest neighbors is identified for each new input. Instead, it determines a suitable value of 'K' for each test sample individually. Initially, the algorithm identifies an optimal value of 'K'. Then, to classify the unlabeled data, 'K' is set equal to the optimal 'K' value of its nearest neighbor in the training dataset. The effectiveness of this approach is evaluated across various datasets to assess its performance.

C. Advantages

KNN is renowned for its simplicity, clarity, and scalability, making it easy to understand and interpret. Its calculation time is minimal, contributing to its efficiency. Moreover, KNN exhibits high predictive accuracy, rendering it effective for large training sets. The classification process in KNN is relatively straightforward compared to other algorithms, involving simple mathematical computations that are easy to grasp. Utilizing basic concepts like Euclidean distance calculation enhances the algorithm's simplicity without resorting to more complex methods like integration or differentiation. Additionally, KNN is adept at handling non-linear data and is versatile, suitable for both classification and regression tasks.

III. Impelemation

The fundamental KNN model is designed with three primary functions: Distance, Vote, and Classify. The Distance function computes the distance between a new data point and unlabeled data. The Vote function constructs a voting table based on the distance matrix. Finally, the Classify function determines the class of the unknown labeled data as the output.

A. *IClassifierKnn Interface*

The provided interface, IClassifierKnn<TIN, TOUT>, outlines the structure for a classifier within the NeoCortexApi namespace. It defines three essential methods. Firstly, the Learn method is responsible for training the classifier model, where it takes input data of type TIN and an array of cells representing the output. This function facilitates the learning process, enabling the classifier to adapt to the provided input-output relationship. Secondly, the GetPredictedInputValue method is designed to predict the input value based on the provided predictive cells. This function is crucial for inference, as it allows the classifier to make predictions or classifications based on the current state of the system. Lastly, the GetPredictedInputValues method extends the predictive capability by enabling the retrieval of multiple predicted input values. It takes an array of cell indices and an optional parameter specifying how many predictions to generate. Overall, this interface provides a versatile framework for implementing classifiers, allowing for learning, inference, and multi-prediction functionalities within the NeoCortexApi.

An internal class named KnnClassifier<TIN, TOUT> is declared, which implements the IClassifierKnn<TIN, TOUT> interface. This integration signifies that the KnnClassifier class conforms to the contract defined by the interface, thereby enforcing the implementation of its methods. The generic type parameters TIN and TOUT indicate that the classifier operates on input data of type TIN and produces output of type TOUT, allowing for flexibility in the types of data it can handle. By implementing the interface, the KnnClassifier class must provide concrete implementations for the Learn, GetPredictedInputValue, and GetPredictedInputValues methods specified in the interface. This ensures consistency and interoperability within the NeoCortexApi namespace, allowing other components to interact with instances of the KnnClassifier class through the common interface without needing to know specific implementation details.

B. *Integration of the KNN*

The integration of the KNN (K-Nearest-Neighbor) Classifier within the Neocortex API framework enables the efficient classification of unlabeled sequences by leveraging a supervised learning approach. MultisequenceLearning and Program classes are already implemented in NeocortexApi. The only class that needs to be modified is the Predictor, see [11]. Then the classifier is trained using a set of labeled sequences, where each sequence is associated with a specific label denoting its class membership. The training data is organized into a dictionary structure, with labels serving as keys and sequences as values. This organized representation facilitates the learning process, allowing the classifier to discern patterns and relationships between sequences and their corresponding labels. Once trained, the classifier can accurately predict the label for new, unclassified sequences based on their similarity to the learned patterns. The prediction process involves identifying the K-nearest neighbors of the unclassified sequence within the training data and assigning the label based on the most frequently occurring label among the nearest neighbors. In the given example, the classifier is tested with an unknown sequence, and the output provides a ranked list of ClassifierResult objects, indicating the closest matches to the unknown sequence. This seamless integration of the KNN Classifier within the Neocortex API empowers users to perform efficient and accurate sequence classification tasks, catering to a wide range of applications across diverse domains.

C. *UNIT TESTING*

For ensuring the reliability and effectiveness of the KNN Classifier integrated into the Neocortex API, a rigorous suite of unit tests is indispensable. These tests serve to verify the functionality and accuracy of the classifier across various scenarios and input configurations. One such unit test focuses on evaluating the training process, where a known dataset is provided to the classifier for model construction. This test assesses the classifier's ability to correctly associate input sequences with their corresponding labels, thereby validating the training procedure's accuracy. Additionally, another unit test concentrates on the prediction process, where an unknown sequence is supplied to the classifier to assess its predictive capability. This test ensures that the classifier accurately predicts the label for the given sequence, thus validating its efficacy in real-world application scenarios. By systematically conducting these unit tests, the Neocortex API team can ascertain the robustness and correctness of the KNN Classifier, affirming its reliability and suitability for a wide range of classification tasks within the API framework.

Conclusion

We commenced our endeavor by conceptualizing a prototype for the KNN algorithm, emphasizing a more generalized approach to accommodate diverse datasets and yield the desired outcomes effectively. Subsequently, leveraging insights gleaned from this prototype, we refined our approach to craft the KNN model with greater specificity, tailored to address the intricacies of the task at hand. This iterative process allowed us to hone the model's capabilities, ensuring its adeptness in handling various data patterns and classifications. Upon achieving satisfactory results with the developed KNN model, our focus shifted towards its seamless integration within the Neocortex API ecosystem. Through meticulous implementation and rigorous testing, we successfully amalgamated the KNN classifier into the API framework, enabling it to seamlessly process streams of data sequences and provide accurate predictions.

The integrated model exhibits a remarkable performance, showcasing an accuracy rate across a majority of input data sequences. This high level of accuracy underscores the efficacy and reliability of the KNN classifier in discerning patterns and making precise classifications within the Neocortex API environment. Moreover, to fortify the robustness of our solution, we meticulously designed and implemented a suite of unit tests to handle specialized cases and edge scenarios. Drawing inspiration from established methodologies, particularly referencing the HTM Classifier, we tailored our unit tests to comprehensively assess the functionality and performance of the KNN model. The culmination of these efforts has yielded highly satisfactory results, reaffirming the efficacy of our approach and the seamless integration of the KNN classifier within the Neocortex

REFERENCES

[1] Mitchell, T. (1997). Machine Learning. McGraw Hill.

[2] Gosavi, A., Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning, Series: Operations Research/Computer Science Interfaces Series, Vol. 25, 2003, Springer.

[3] I. Chickering, D.M. (2002). Optimal Structure Identification with Greedy Search. Journal of Machine Learning Research, Vol. 3, pp 507–554.

[4] Numenta, Hierarchical Temporal Memory including HTM Cortical Learning Algorithms, HTM CLA white paper - VERSION 0.2.1, SEPTEMBER 12, 2011.

[5] A.J. PereaI, J.E. MeroñoII, M.J. AguileraI, Application of Numenta® Hierarchical Temporal Memory for Land-Use Classification, South African Journal of Science, ISSN 0038-2353, vol.105 n.9-10 Pretoria Sep./Oct. 2009.

[6] P. WiraBuana, S. Jannet D.R.M., and I. Ketut Gede Darma Putra, “Combination of K-Nearest Neighbour and K-Means based on Term Re-weighting for Classify Indonesian News,” Int. J. Comput. Appl., vol. 50, no. 11, pp. 37–42, Jul. 2012.

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

[8] D. Wettschereck and D. Thomas G., “Locally adaptive nearest neighbour algorithms,” Adv. Neural Inf. Process. Syst., pg. 184–186, 1994.

[9] Han EH.., Karypis G., Kumar V. (2001) “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 Science, vol 2035. Springer, Berlin, Heidelberg.

[10] Shengyi Jiang,Guansong Pang,Meiling Wu,Limin Kuang, “An improved K-nearest-neighbour algorithm for text categorization”, Expert Systems with Applications,Elsevier(2012)

[11] https://github.com/UniversityOfAppliedSciencesFrankfurt/se-cloud-2022-2023/tree/Team_Mariglen_Kejdjon/MySEProject