*Implement the KNN Classifier*

Kashif Hussain  
kashif.hussain@stud.fra-uas.de

Naila Shaheen  
naila.shaheen@stud.fra-uas.de

Fahad Jabbar  
fahad.jabbar@stud.fra-uas.de

*Abstract—* ***This paper presents a K-Nearest Neighbors (KNN) classifier designed for the Neocortex API; a software toolkit inspired by neocortical information processing. This approach deviates from conventional KNN implementations by leveraging Neocortex API 's capabilities for processing sequential data, similar to the neocortex. The proposed method integrates spatial pooling for dimensionality reduction and hierarchical temporal memory (HTM) for temporal pattern recognition, enabling efficient classification of sequential data streams. This work bridges the gap between machine learning and neuromorphic computing by introducing a KNN classifier within a neocortex-inspired framework, offering a novel perspective on KNN classification and paving the way for further exploration at the intersection of these fields.***

*Keywords—Machine Learning; K-Nearest Neighbors; Neocortex API;* *Hierarchical temporal memory.*

# Introduction

Our brains are remarkable pattern recognition machines, and machine learning draws inspiration from this biological marvel. Traditional classification algorithms often treat data points as isolated entities. This is unlike the human neocortex, where interconnected neurons process information in a hierarchical fashion. The neocortex excels at recognizing not just individual features, but also complex relationships between them.

HTM seeks to emulate the structure and function of the neocortex by employing a layered architecture. The Spatial Pooler, akin to initial cortical processing, extracts feature from sensory data, representing them sparsely. This mirrors the neocortex's ability to identify essential patterns, such as edges or shapes, while disregarding irrelevant details. However, the neocortex doesn't stop at feature extraction; it integrates these features over time to build a coherent understanding. HTM's Temporal Memory serves this purpose by analyzing sequences of representations, allowing the system to learn temporal relationships and predict future patterns. Unlike traditional classification algorithms, HTM comprehends context and sequence, offering a deeper understanding of data beyond mere categorization. Our goal is to create a reliable KNN algorithm and integrate it with the Neocortex API. This API is specifically built to work with HTM. It excels at understanding patterns in dynamic datasets, such as those found in environmental sensors, multimedia, and time-series data. Combining HTM with KNN is especially useful when dealing with data that has complex temporal structures, where traditional classification methods fall short.

**Supervised and Unsupervised Machine Learning**

Supervised learning entails the utilization of labeled data to establish a mapping function that correlates input features *x* with the output variable *y*, represented as *y* = *f*(*X*,*θ*). Through this process, supervised learning aims to discern patterns and relationships within the data to facilitate accurate predictions or classifications. It also involves the training a model to predict discrete or categorical output variables *y*. On the other hand, unsupervised learning operates on unlabeled data, seeking to uncover inherent structures and patterns without predefined output variables. Unsupervised learning techniques, including clustering, anomaly detection, and latent variable mixture models, are pivotal in extracting meaningful insights from unannotated datasets (Rajoub, 2020).

**Classification** is a machine learning technique used to predict group membership for data instances. It involves assigning predefined categories or classes to input data based on their features or attributes. The primary objective of classification is to develop models that can accurately distinguish between different classes within a dataset, enabling automated decision-making and pattern recognition. To achieve this, classification algorithms analyze the characteristics of labeled training data to learn patterns and relationships that can be generalized to classify new, unseen instances. These algorithms employ various approaches, such as statistical methods, decision trees, support vector machines, and neural networks, each with its own advantages and suitability for different types of data and applications. For instance, in a classification scenario, the objective is to assign a given observation to one of several predefined classes {*C*1​,*C*2​,…,*Ck*​}, based on learned patterns from the labeled data (Soofi & Awan, 2017).

**Regression** is a machine learning technique used to predict continuous numerical values based on input features. Unlike classification, where the goal is to classify data into predefined categories, regression aims to estimate the relationship between independent variables and a dependent variable. This relationship is typically represented by a mathematical function that predicts the value of the dependent variable given the values of the independent variables. Regression models are trained using labeled data, where the target variable is continuous and can take any value within a range. Common types of regression include linear regression, polynomial regression, ridge regression, and lasso regression, each suited to different types of data and modeling scenarios. Linear regression, for example, assumes a linear relationship between the independent and dependent variables, while polynomial regression allows for more complex, nonlinear relationships. For instance, regression models may be employed to predict phenomena like CO2 emissions on specific dates, leveraging historical data to infer future trends and patterns (Maulud & Abdulazeez, 2020).

**K-Nearest Neighbor (KNN)**

In the KNN technique, nearest neighbor is measured with respect to value of k, which define how many nearest neighbors need to examine to describe the class of a sample data point. To balance the overfitting and underfitting, parameterization of the k value is essential to the KNN, because of its adaptability, practitioners can customize the model to fit certain datasets and classification needs. Additionally, the instance-based learning nature of KNN, where classification decisions are made locally, makes it well-suited for scenarios with large or dynamic datasets (Nikhath et al., n.d.). Its success relies on selecting the best distance metric, distance is calculated between sample points and all training points and the point with smallest distance is known as nearest neighbor. Despite its simplicity, it's robust and versatile, making it a popular choice for classification across diverse domains. Mostly it relies on Euclidean distances as a default metric (Farou et al., 2024).

# **Literature Review**

# **Literature Review**

In this section, we'll take a closer look at the k-nearest neighbor (KNN) algorithm, exploring its workings and applications. Our aim is to understand KNN, and it's tasks like pattern recognition and classification. We will examine KNN from numerous study perspectives and analyze the integration of HTM with KNN in detail, offering comprehensive insights into the procedure.

For classification purposes, a class label is determined by a majority vote, where the label most frequently observed among neighboring data points is assigned. This terminology distinction arises because "majority voting" necessitates a majority of over 50% of the vote, typically applicable in binary classification scenarios. In cases involving multiple classes, such as four categories, reaching a 50% threshold may not be essential for class determination; assigning a label based on a vote exceeding 25% is sufficient (Raschka, 2018).

The study by Shichao Zhang (Zhang & Li, 2023) on K-Nearest Neighbor (KNN) classification shows that determining the optimal value for K and conducting nearest neighbor queries are critical tasks. Regarding the nearest neighbor, various distance measurement functions like Euclidean distance, Mahalanobis distance, Manhattan distance, and angle cosine distance are employed. However, the method for calculating K value remains a challenge. Currently, it's predominantly achieved through expert settings or cross-validation techniques. However, these methods do not address the inherent issue of K value calculation because all test data share the same K value. Selecting a small K value can lead to overfitting, while opting for a large K value may increase the approximation error of learning. To address this, Zhang et al. proposed reconstructing the relation matrix and introduced a method for optimal K value computation, aiming to mitigate these challenges.

The research by Sun and Huang (Sun & Huang, 2010) introduced an adaptive K-nearest neighbor algorithm, which operates by initially computing the optimal K value for each training data point based on the Euclidean distance metric. Subsequently, it identifies the nearest neighbor of the test data and employs the optimal K value of this nearest neighbor as the optimal K value for the test data. Ultimately, the KNN classification process is executed following the majority rule principle.

Liu et al. (Liu et al., 2018)introduced a weighted K-nearest neighbor (KNN) algorithm that adapts the K value for each test data point based on local characteristics derived from surrounding training data. This dynamic approach enables the algorithm to effectively adjust to varying densities and complexities within the feature space. Additionally, the algorithm computes weights for neighboring data points relative to their distances, providing a mechanism to emphasize the influence of nearby instances while mitigating the impact of outliers. Subsequently, classification is performed using the majority rule principle, considering the weighted contributions of neighboring points. This adaptive and weighted approach enhances the robustness and accuracy of KNN classification, particularly in scenarios where data distribution is non-uniform or where optimal K values vary across different regions of the feature space.

In the era of modern healthcare, the analysis of medical images, such as MRI brain scans, has become a crucial task for diagnosing diseases and monitoring patient health. However, interpreting these images manually is time-consuming and prone to errors. To address this challenge, researchers are exploring the integration of K-nearest neighbor (KNN) algorithms with artificial neural networks (ANNs). ANNs, particularly adept at recognizing patterns in complex data like MRI images, offer a promising approach to automate image analysis tasks. The speed and accuracy of

identifying conditions from MRI scans can be increased by utilizing the combined capabilities of KNN and ANNs, which will eventually improve patient outcomes and healthcare efficiency. With advancements in algorithmic techniques, KNN is increasingly recognized as a valuable tool for medical image classification and analysis (Anna University. Department of Information Technology. & Institute of Electrical and Electronics Engineers. Madras Section., 2011).

Recent research emphasizes the importance of selecting an optimal K value for accurate classification, with adaptive techniques proposed to determine K based on local data characteristics. Approaches such as weighted KNN and adaptive algorithms enhance classification robustness by adjusting to varying data complexities. Overall, advancements in KNN techniques underscore its significance as a valuable tool for pattern recognition and classification tasks across different domains, contributing to improved accuracy and efficiency in diverse applications.

# **Theoretical Background**

The k-nearest neighbor (KNN) algorithm, which is a supervised machine learning algorithm, it uses training data and a predetermined k value to identify the k closest data points through distance calculations. When these k data points belong to different classes, the algorithm predicts the class of the unknown data to align with the majority class. This approach, which is based on statistical principles, aids in making informed decisions based on the collective attributes of neighboring data points. The overall process of KNN working principle is explained below.

**1. Distance Metrics**

Distance metrics are a fundamental component of the k-nearest neighbor (KNN) algorithm, essential for determining the similarity between data points in classification tasks. KNN operates by calculating distances between a new data point and existing data points in the training dataset. This process entails evaluating the dissimilarity or similarity using various distance metrics, such as Euclidean distance, Manhattan distance, and cosine similarity (Chomboon et al., 2015)

**1. Minkowski distance function**

Minkowski distance (Boelens & Tchelepi, 2020) function is a vector space. It measures the geometric distance among two inputs using *p,* which is a variable scaling factor written in equation.

*MK* =

It can be deployed to guess the distance between two points by using diverse values of scaling factor *p*. For instance, Manhattan distance is obtained when *p*=1; Euclidean distance is obtained when *p*=2.

**2. Mahalanobis distance function**

The Mahalanobis metrics (Zeng et al., 2020) DM can be defined as the distance between an observation and the center of each data group in an *n*-dimensional space defined by *n* variables and their covariance. For some classifications, it is verified more productive (Taunk et al., 2019). The problem of scale and correlation associated with EUD function has been suitably talked by Mahalanobis distance. It does not depend on the unit of measurements of the dataset; it finds the distance between two groups (Jaiyen & Sornsuwit, 2019). It varies from EUD in that it takes into account the correlations of the data set and is scale-invariant (Pilinszki-Nagy & Gyires-Tóth, 2020). It is also called statistical distance; hence it is a comparative benefit. Let we have two groups with means 𝑋𝑗 and 𝑋𝑘, 𝐷𝑀, is given by equation

*DM*

Where is a converse pooled covariance matrix S among 𝑎𝑛𝑑 . This matrix is figured by using a weighted average of covariance matrices of two groups.

**3. Euclidean distance function**

The Euclidean distance (Maxim et al., 2020) EUD is the distance between two points that are linked by a straight line. If and in the space of n dimensions (You et al., 2019), Rectilinear distance, city block distance, or taxicab metrics are the name of EUD.

*EUD*

Euclidean distance, a core concept in mathematics, measures the straight-line distance between two points in space. It's calculated as the square root of the sum of the squared differences between corresponding coordinates. In machine learning, Euclidean distance is frequently used in algorithms like K-nearest neighbors (KNN) to determine the similarity between data points, aiding in tasks such as classification and clustering.

(1)  
Where:

P = , , …..., represent the coordinates of point P

Q = , , …..., represent the coordinates of point Q

*n* denotes the number of dimensions in the space

**3. Manhanttan distance function**

Manhattan distance function (Jiang et al., 2019) MN calculates the distance among two points. It is the line section’s projection point’s addition that occurs between the points. These points belong to the coordinate axes. If and in the space of n-dimensions.

*MN*

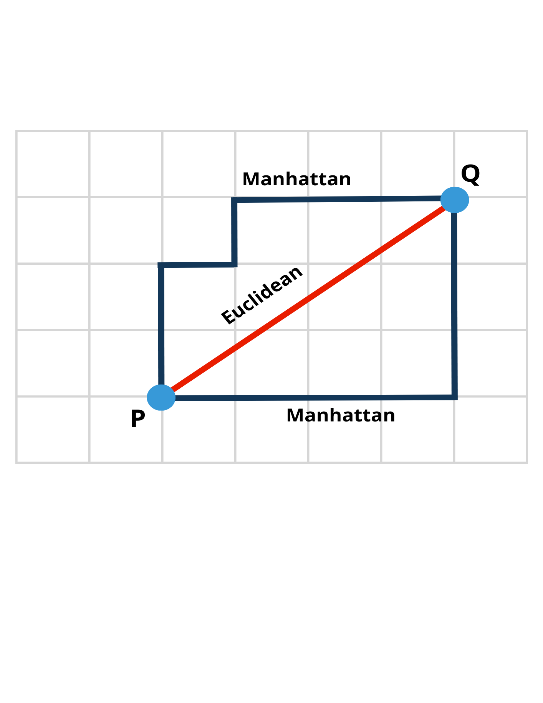
Manhattan Distance calculates the distance between two points by summing the absolute differences of their coordinates. Unlike Euclidean Distance, it evaluates movement along grid-like paths, akin to navigating city blocks in a taxi. This distance is particularly useful in scenarios where movement is constrained to horizontal and vertical paths, such as grid-based environments or city maps.

Where:

P = , , …..., represent the coordinates of point P

Q = , , …..., represent the coordinates of point Q

*n* denotes the number of dimensions in the space.

Here is a graphical representation showing both Euclidean and Manhattan distance methods, which visually demonstrate their respective approaches to measuring distance between two points.

*Figure 1: Comparison between Euclidean and Manhattan distance metrics*

**2. Value of K**

The value of k in kNN classification plays a critical role in determining the model's performance and generalization ability. Selecting the appropriate value of k involves a trade-off between bias and variance in the model. A small value of k (e.g., k=1) can lead to a more flexible decision boundary but may be sensitive to noise and outliers, potentially resulting in overfitting. On the other hand, a large value of k (e.g., k>10) can smooth out the decision boundary and reduce overfitting but may lead to underfitting if the value is too large for the dataset. Choosing an optimal value of k, often through techniques like cross-validation and considering domain-specific knowledge, is essential to strike a balance between capturing the underlying patterns in the data and avoiding overfitting or underfitting, ultimately improving the classifier's predictive performance (Zhang et al., 2017).

**3. Voting in KNN:**

In the k-nearest neighbors (kNN) algorithm, voting is a crucial step in the classification process. After identifying the k nearest neighbors to a given data point, the algorithm assigns a class label to the data point based on a majority or weighted vote from its nearest neighbors.

Majority Voting: In the majority voting scheme, the class label assigned to a data point is determined by the most common class among its k nearest neighbors. This simple approach is effective when all neighbors have an equal vote weight.

Weighted Voting: Weighted voting assigns different weights to the votes of the nearest neighbors based on factors such as their distance to the data point. This allows closer neighbors to have a stronger influence on the classification decision. Weighted kNN variants like Distance-Weighted kNN (WKNN) and Dual-Weighted kNN (DWKNN) have been proposed to improve the performance of the traditional kNN algorithm by incorporating weighted voting mechanisms (Mateos-García et al., n.d.).

**Hierarchical temporal memory and Neocortex API**

Hierarchical Temporal Memory (HTM) is a machine learning model developed based on the principles of the neocortex, the part of the brain responsible for higher cognitive functions. HTM is designed to mimic the structure and function of the neocortex, particularly focusing on learning sequences and patterns in data. The HTM model consists of different components such as the Spatial Pooler, Temporal Memory, and SDR Encoder, which work together to process and learn from input data. The Spatial Pooler is responsible for creating Sparse Distributed Representations (SDRs) of input patterns, while the Temporal Memory component focuses on learning temporal sequences and patterns. By utilizing these components, HTM can recognize and predict sequences with robustness and efficiency, making it a powerful tool for various machine learning tasks. [14]

# References