**K-Nearest Neighbors Worksheet**

Today, we are learning about classification, a family of machine learning models that help us categorize samples into groups. Classification is similar to clustering in that we try to find the closest group to match samples to, but unlike clustering, we provide the model with training data where a human has already distinguished one group from another.

1. Based on what you know so far, would you  
   consider classification to be a method of  
   supervised or unsupervised learning?

Think about how your email decides which messages are spam and which ones are important. This is an example of classification. The algorithm we will explore today is called K-Nearest Neighbors (KNN).

The K-Nearest Neighbors (KNN) algorithm is a simple, yet powerful, classification method that relies on similarity between data points. Unlike K-means or DBSCAN, KNN does not attempt to create clusters from scratch. Instead, it classifies or predicts a data point’s value based on the majority label or average of its nearest neighbors. There are forms of KNN that can be used for regression, but in this module, we will focus on classification problems.

Consider the following dataset that is being analyzed with K-Nearest Neighbors:

|  |  |  |
| --- | --- | --- |
| **X1** | **X2** | **Y (class)** |
| 3 | 4 | A |
| 1 | 2 | A |
| 2 | 8 | A |
| 6 | 5 | B |
| 7 | 8 | B |
| 5 | 6 | B |
| 1 | 3 | A |

1. On the next page, neatly plot these points with X1 being along the x-axis and X2 being the y-axis (label the axis accordingly). Use different colors or shapes to distinguish one class from another.

10

10

1. Add a new, unknown sample at (3,5). This sample  
   does not yet belong to a class. Visually estimate  
   which class it should belong to. Which class would  
   you assign it to?

KNN relies on distance measurements to determine the closest neighbors to a given sample. One of the most common distance metric is Euclidean Distance, although there are many other distance functions. For this exercise, we will be using Manhattan Distance.

1. What is the formula for Manhattan Distance?

Consider using K = 3 (meaning we consider the 3 closest neighbors).

1. Use the table on the next page to compute the Euclidean distance from (3,5) to all other samples in the dataset. Identify the three closest points. Then assign the most common class among these neighbors to the new sample.

|  |  |  |  |
| --- | --- | --- | --- |
| Sample | Manhattan Distance to (3, 5) | Distance Ranking  (1, 2, 3, …, 7) | Sample Class |
| (3, 4) |  |  |  |
| (1, 2) |  |  |  |
| (2, 8) |  |  |  |
| (6, 5) |  |  |  |
| (7, 8) |  |  |  |
| (5, 6) |  |  |  |
| (1, 3) |  |  |  |

1. Which three points are the closest?  
     
   What is the majority class?

K is a crucial parameter in K-Nearest Neighbors. A small K (e.g., K=1) can be sensitive to noise, while a large K (e.g., K = 10) can over-generalize the decision boundary.

1. Set K = 1. What class does sample (3, 5) belong to?
2. Set K = 5. What class does sample (3, 5) belong to?
3. Go back to K = 3 but use Euclidean Distance (just measuring with a ruler).

|  |  |  |  |
| --- | --- | --- | --- |
| Sample | Euclidean Distance to (3, 5) | Distance Ranking  (1, 2, 3, …, 7) | Sample Class |
| (3, 4) |  |  |  |
| (1, 2) |  |  |  |
| (2, 8) |  |  |  |
| (6, 5) |  |  |  |
| (7, 8) |  |  |  |
| (5, 6) |  |  |  |
| (1, 3) |  |  |  |

KNN is considered a lazy learning algorithm, meaning it does not explicitly train a model. Instead, it stores all of the training data and classifies new points at runtime, when needed. It requires no training, meaning it is easy to implement. However, it has some downsides:

Think through the implications of KNN’s lazy algorithm.

1. What are the advantages of KNN?
2. What are some downsides of KNN?

Let’s start thinking about how we might implement KNN in code. Consider a class-based design.

1. What data would you need to store inside of the class?
2. Identify the main functions that you would implement.
3. Write some pseudocode outlining how these functions interact to classify a new sample.

Here is some AI generated pseudocode.

**class** **KNN**:

**def** **\_\_init\_\_**(self, k):  
 self.k = k  
 self.data = []

**def** **fit**(self, X, y):  
 """ Store training data. """  
 self.data = list(zip(X, y))

**def** **predict**(self, sample):  
 """ Find the k-nearest neighbors and assign a label  
 based on majority vote. """  
 distances = [(**euclidean\_distance**(sample, point), label)  
 for point, label in self.data]  
 distances.sort()  
 neighbors = distances[:self.k]  
 **return** **majority\_vote**(neighbors)

1. How would you change this implementation to work faster with large datasets?
2. What would you add to allow the user to choose from a variety of distance functions?