KNN

(k nearest neighbors)

Marc Jalkh, Maria Olano, Shrishti Thakur

Table Of Contents

Objectives

Purpose

- KNN & machine learning
- Importance in AI
- Python implementation
- Can be matched with OOP

Learning Outcomes

- Basic Machine learning process
- Foundations of the KNN algortihm
- Data handling and classification

Course Applications

- OOP
- Pandas and file handling
- Pseudocodes and logic

Collaboration

- Division of fiunctionalities
- Iteration reports
- Multitasking
- Git situation

Presentation

- Clarity & Conciseness
- Try our best to maintain interest
- Explanation efficiency

Team Timeline

Note: *Our team met approximately once or twice at the occasion of every iteration, completing its requirements in a single sitting . So, all project was done in collabaration with evey team member, hence the timeline's resttrictions solely to iteration requirement*

Iteration 1 (02/18)

- Agreeing on Topic
- Choosing teamates

Iteration (03/17)

- Starting code
- Choosing algoritm
- 2 options plotting/distance

Iteration 3 (03/25)

- Finishing all code aspects
- Agreeing on Data type
- Agreeing on Data handling method

Iteration 4 (03/31)

- Presentation prep
- Demonstrations

Tool list

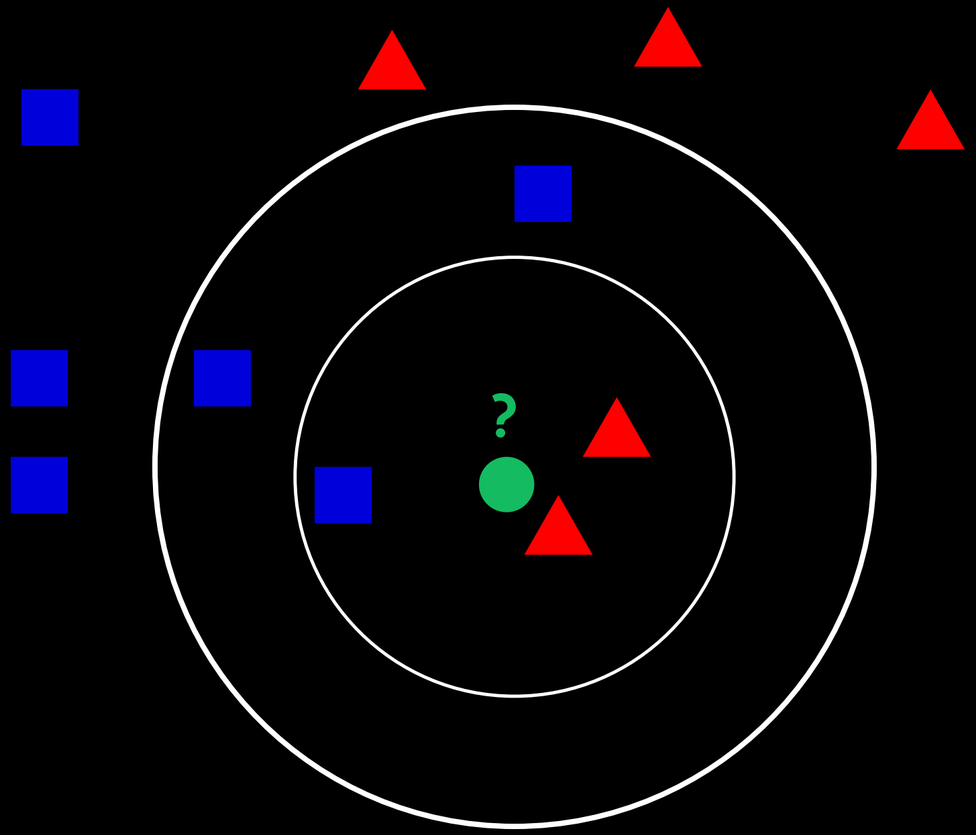
Python

- Python as only necessary software
- Includes enough tools for basic KNN implementation
- Libraries and Data handling

CSV & Pandas

- Python library
- Imports CSV table
- Table as database on which KNN will be applied

k-nearest neighbors (KNN)



Developed by Evelyn Fix and Joseph Hodge in 1995 as research for the US military.

Based on the assumption: “similar objects tend to be found near each other.”

- Retention of Training Data: KNN stores all training data for future reference.
- Similar Data Identification: It compares new data with stored data, seeking similarities through distance measurements.
- Selection of Neighbors: The algorithm selects the 'k' closest data points, where 'k' is a user-defined parameter.
- Classification Method: KNN determines predictions by assessing the most common class among the selected 'k' neighbors.

Stores data and uses it directly to make predictions.
All about finding nearest neighbors based on how you define closeness.

k-nearest neighbors (KNN)

Implementation

- Measure the distance between the current data point and all points in the training set to find the 'k' nearest neighbors.
- Sort the distances and select the 'k' smallest ones to pinpoint the nearest neighbors to the data point.
- Combine the class labels of the 'k' nearest neighbors to predict the class of the data point.

Considerations: Nature of data, scale of features, dataset size, data distribution.

Today's Use

- Classification tasks: spam and anomaly detection, and customer segmentation.
- Recommendation systems: collaborative filtering and content-based filtering.
- Regression Tasks: predictive analytics and demand forecasting.
- Image retrieval and classification
- Healthcare and Medical Diagnosis
- Robotics and AI

Prompt

KNN & cantina case

KNN Prerquisites

- Based on Han Solo's table
- Data set, of individuals with known attributes
- Classifiable ? Yes.
- The aliens and their variations of attributes
- A single to be determined "Dangerous" classification
- All data needed is given in the cantina case prompt

Note: *Legend Known & To be determined*

Color	Height	Weight	Eyes in pairs	Dangerous
Yellow	Average	Light	Single	No
Yellow	Tall	Normal	Pairs	Yes
Green	Short	Normal	Pairs	Yes
Yellow	Short	Normal	Single	No
Red	Average	Heavy	Single	No
Green	Tall	Heavy	Single	Yes
Green	Average	Heavy	Single	Yes
Yellow	Short	Light	Pairs	Yes

Setup 1: Prerequisites

- Classification norm as Python dictionary
- Pandas and CSV file reading
- Separation known & to be determined attributes
- Classification of known attributes into points based on norm & insertion into a list
- Attributes from non/numeric to classified coordinates
- Distance function for later use (basic math formula)

Note: *Setup in main( ) independent from class*

Setup 2: Class requirements

Required arguments for proper KNN application:

- Classification dictionary
- Points with coordinates list (tuple list)
- To be determined attribute list
- User input of new alien variation of each attribute

Based on his known attributes and cantina points, unknown attribute will be determined

Range of search determined by choosing k

Color	Height	Weight	Eyes	CLassification
Yellow	Short	Light	Single	1
Green	Average	Normal	Pairs	2
Red	Tall	Heavy	N/A	3

Step 1 & 2: Classification & Distance

Method 1: Classification

- Use the classification dictionary taken as argument and classify user attribute input based on it.
- Will obtain a "*new point*" coordinates as tuple

Method 2: Distance calculations with distance function

- Compute distances from each point to *new point*
- Into nested list: *[[d0, p0], [d1, p1], . . . , [di, pi]*

Note: *Important to keep track of both for next method*

Step 3: Sorting & nearest neighbors

Method 3: Bubble sort, k taken as user input of argument

- Nested list bubble sort k(k − 1)/2.
- Closest from n*ew point* to furthest
- Cut list to k(th) index, hence k nearest neighbors
- Discard distances

Note: *k as range of search based on proximity*

Step 4: Declassification

Method 4:

- Traceback selected points relating them to their Dangerous attribute based on list search and indexes
- Insert variations into a list such as: ["Yes","Yes","No"...]

Note: *This method considers every cantina alien unique to be able to traceback Dangerous attribute*

Step 5: Voting

Method 5:

- Based on abundance of missing attribute variations in list previously extracted
- Count → variation with highest abundance between the k nearest neighbors is the one assigned to the *new point*
- Hence predicting behavior using KNN