K-NEAREST NEIGHBOR CLASSIFIER

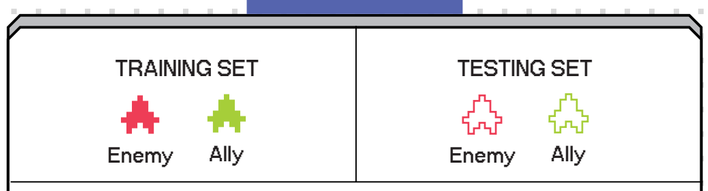
Hi recruit IX 209, welcome to the Main Control Tower!

Your role here will be to monitor the presence and position of enemies on the radar and to communicate them in real time to our allies' ships.

Sometimes **unidentified** ships appear on the radar, but the system can't classify them due to a lack of signal or communication.

In those cases, you'll have to rely on this machine, that makes use of **KNN Classifier**—an algorithm that, based upon the characteristics of a **certain number** (**K**) of **known elements** that are the **most close-by** (**Nearest Neighbor**) to new unknown data, **predicts the category** of it (**Classifier**).

Dataset

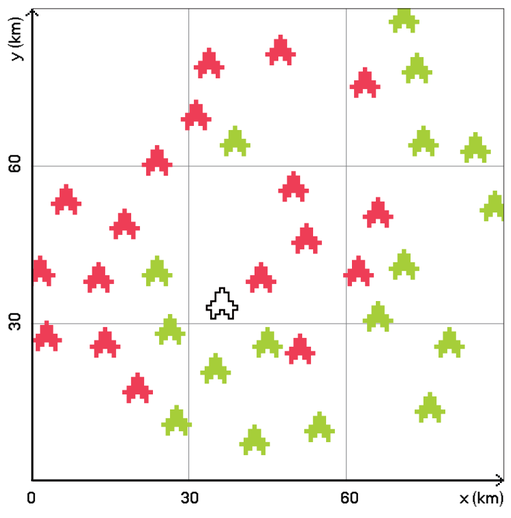


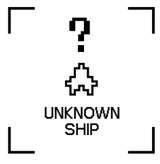
We always start from a **dataset**. In our case, it's usually constituted by the **locations**—consisting of two spatial coordinates in a Cartesian plane—of the known allies and enemies in certain moments of time, since they often tend to group along themselves. Then, we split the dataset into a **training** and a **testing set**: the first one is used to calculate the predictions, while the second one to assess its accuracy. In this case, the training set contains **75%** of all the known locations, which is enough to tell how the ships are distributed.

Start

Step 1

Setting the parameters

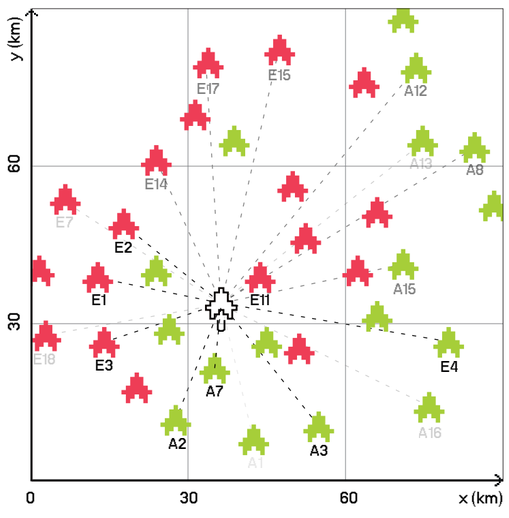


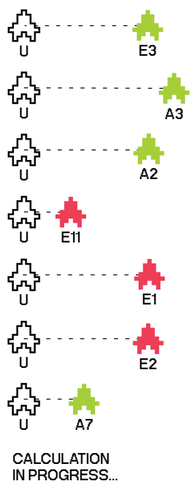


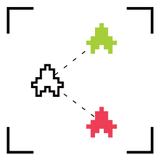
Warning! An **unknown ship** appeared on the map and you have to figure out whether it is an enemy or an ally. The **training set** will be used to make the **prediction**. Before the algorithm can start computing, you have to **initialize it** by setting the **K value** and a **distance formula**. Let's settle for a K value of 4 and the Euclidean distance.

Step 2

Computing distances



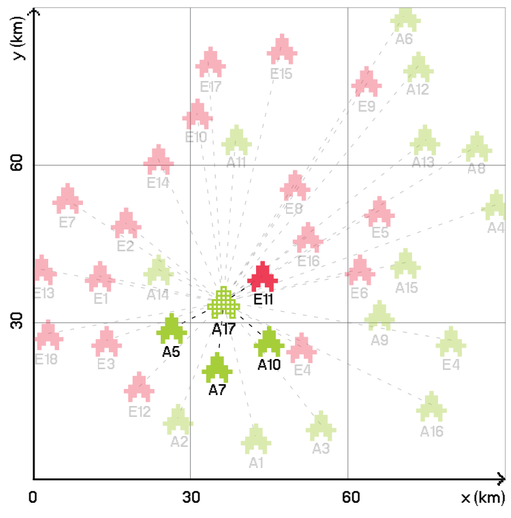


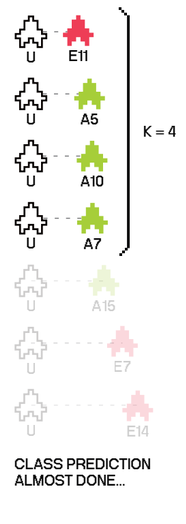


After doing so, the algorithm **computes the distances** between the position of the unknown ship and each of the known ones via the Euclidean distance, a fast and accurate way to measure the length of a line segment between points in a space using the Pythagorean theorem. All the distances will then be stored in a **list**.

Step 3

Sorting and assigning





Then, the algorithm orders all the distances from nearest to farthest and **selects the top k with the shortest distances**. The k value defines **how many neighbors** will be checked to predict the class of the unknown ship. Finally, the algorithm **counts the most frequent class** out of those and **assigns it** to the unknown ship. Fortunately, it seems to be an ally!

How to find the best K value with the test-train method

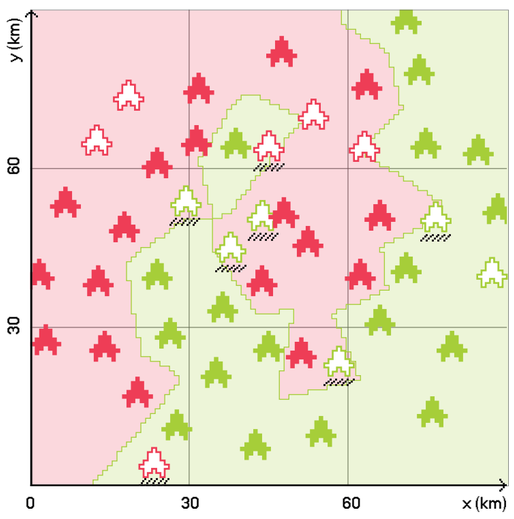
There are no shortcuts: the **right K value** changes along with the given distributions of known ships and you must carry out a **testing phase** to find it every time. This process can be carried out with many different techniques, but let's focus on the **test-train split method**.

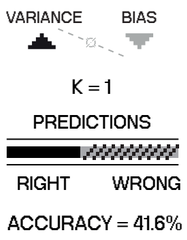
Since this machine relies on your supervision to be reliable, you have to be sure that the K value you use drives accurate and consistent results, thus preventing two significant issues: **overfitting and underfitting**.

Split the dataset › Load the **training set** › Load the testing set as **unclassified data** › For **every k** you want to test: **Predict the classes of the testing dataset via the KNN Classifier** › **Compare** the predictions with the **actual classes** of the **testing set** › **Calculate** the accuracy percentage based on the amount of **correct predictions** › Select the K value that corresponds to the **highest accuracy** percentage

Overfitting

Too tight

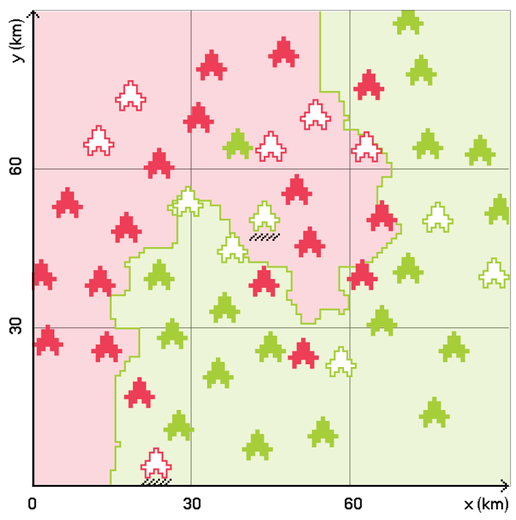


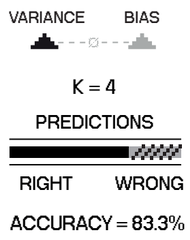


Overfitting happens when the value of K is too low. The predictions will be based on too few known locations of the training set, resulting in a **squiggly boundary** between classes. The algorithm will be **too strict** and **incapable of reacting consistently** to the way ships are going to appear.

Ideal Balance

Best fit

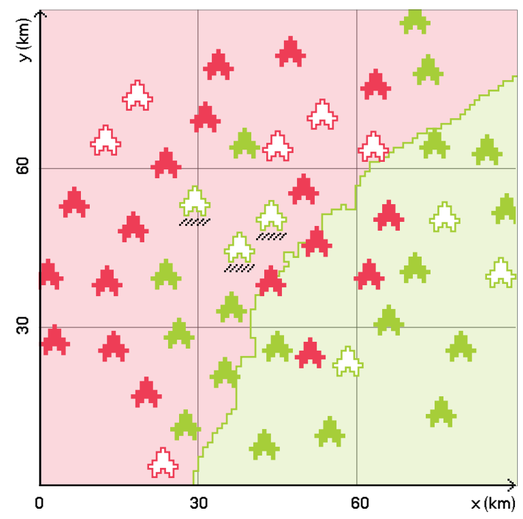


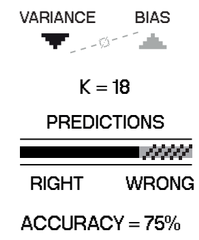


The optimal K is found between those two cases. You'll be able to tell that you found it—or at least that you are close to it—when **the boundary** between classes starts to **maintain a balance** between complexity and flexibility, resulting in an **appropriate representation** of the way ships are going to appear.

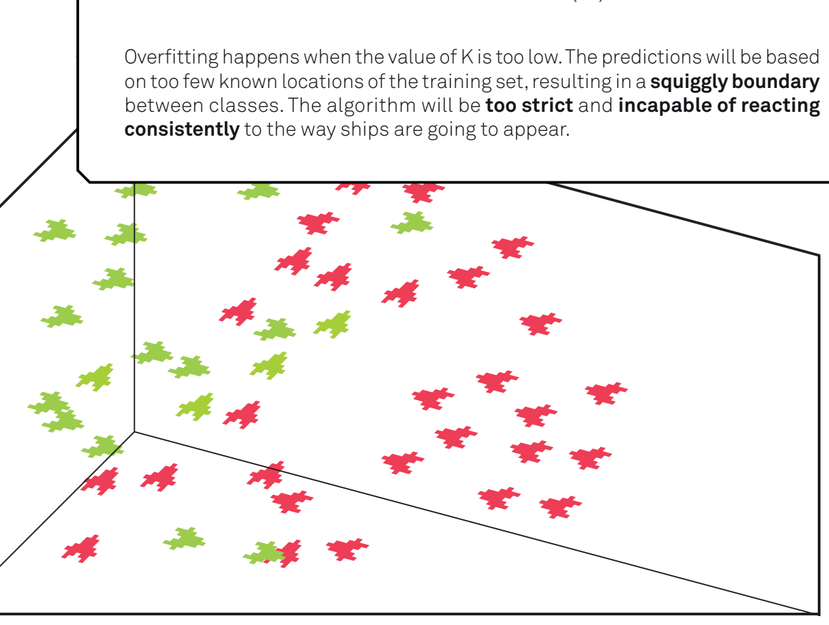
Underfitting

Too loose





Underfitting happens when the value of **k is too high**. In this case, the boundaries between classes will be much smoother and more approximated, since the predictions will be based on a large amount of known locations of the training set. The algorithm will be **consistent**, but also **consistently wrong**.



Wait! Just a few last things before I let you go:

Features

To help us strategize **in complex scenarios**, you could **add** even **other characteristics** (**features**) to the dataset, like the direction or the size of the known ships. To do this, the values must be made **comparable** by transforming them into ranges of dimensionless values via **standardization**.

Distances

Distance formulas can **influence the performance of the algorithm**. The most commonly used outside of the Euclidian distance are the Manhattan, Minkowski and Hamming distances, which can be used for **different feature types**.

But remember!

THE MORE FEATURES YOU ADD, THE MORE YOU INCREASE THE RISK OF CLASSIFICATION ERRORS AND THE TIME TO CALCULATE THE PREDICTIONS. BUT YOU DON'T HAVE TO WORRY ABOUT IT, YOU WILL LEARN IN THE FIELD. GOOD LUCK!

VISUAL EXPLANATIONS OF STATISTICAL METHODS

k-nearest neighbor classifier

AUTHORS

Andrea Benedetto
Simone Cerea
Camilla Guerci
Surabhi Gupta
Haleh Nobar
Alessandro Pedriali
Yousef Taffal

FACULTY

Michele Mauri
Ángeles Briones
Gabriele Colombo
Simone Vantini
Salvatore Zingale

TEACHING ASSISTANTS

Elena Aversa
Andrea Benedetti
Tommaso Elli
Beatrice Gobbo
Arianna Bellantuono

Final Synthesis Design Studio
Sect. C3

LM in Communication Design
A.A. 2022/2023

DEN-SITY GN+

POLITECNICO MILANO 1863
SCHOOL OF DESIGN