POLITECNICO DI TORINO

-

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

M.Sc. in Mechatronic Engineering

Master Thesis

Machine-learning-based techniques for the complete coordination prediction of astrobots swarms

Supervisors:
Dott. Matin Macktoobian
Prof. Denis Gillet
Prof. Marcello Chiaberge

Candidate:
Francesco Basciani

For my family,
for my friends,
and for Giulio Regeni

Contents

Abstract

Astrobots are robotic manipulators present in the latest generation telescopes. Their role is to coordinate optical fibers in specific positions of a focal plane, so that it is possible to get astronomical information of a desired observation. The crowded arrangement of these manipulators inside the focal plane of a telescopes makes the coordination process so much complicated that it is not always possible for a single astrobot to reach its target configuration. Therefore the convergence verification before the execution of a coordination is of particular interest. However a formal method for this purpose has not yet been found in the past.
The main objective of this thesis is to develop formal methods for the prediction of convergence of astrobots swarm using some machine learning techniques. In particular two prediction algorithms have been developed, a KNN-based algorithm and an SVM-based algorithm. Furthermore, a possible approach to solving the problem is proposed which exploits the use of a convolutional neural network.

Chapter 1

Introduction

In order to obtain informations from the electromagnetic waves emanated from cosmological objects, the latest generation telescopes are equipped with a series of optical fibers which are used in several spectroscopic surveys. Spectroscopic data play an essential role in the study of the dark matter. The optical fibers are then needed to realize this kind of observations inside telescopes. Within the focal plane of a telescope, each fiber must be moved according to the position of the observation of a specific cosmological object. Each fiber is then attached to a positioner to be automatically coordinated. The control algorithm that coordinates the movement of each positioner has already been developed and exploits a decentralized navigation function. The coordination challenge is given by the fact that inside a focal plane thousands of fibers are often present that need to be coordinated at the same time and avoiding possible collisions.
Unfortunately, the achievement of the desired position by a given positioner is not always guaranteed. This is because the developed control algorithm is subject to some local minimums which cause the positioner to stumble in deadlocks. The main objective of this master thesis is to develop an algorithm which is able to predict the convergence to the respective target positions of a swarm of positioners (which from now on will be called astrobots), given their positions inside a focal plane, and their parities (the way the astrobots approaches the target, which can be clockwise or counterclockwise).
In the continuation, we use bold symbols to indicate matrices or vectors, while we use regular symbols to indicate scalars or sets.

1.1 Description of focal plane

The standard design for a focal plane consists of a collection of identical astrobots distributed over an hexagonal array (See fig 1.1). Every astrobot has an assigned target. The focal plane could be entirely covered by astrobots, and each of them moves a fibre toward the desired location, that is within the patrol disc of the astrobot. Since it is necessary that any point of the focal plane is accessible by a fiber, there will be regions of the focal plane where the workspaces of adjacent astrobots will overlap and consequently in these zones there is a risk of collision. The problem of target assignment and collision avoidance are challenges that have already been tackled and studied in the design of such massively parallel fibre-fed spectrographs [1].

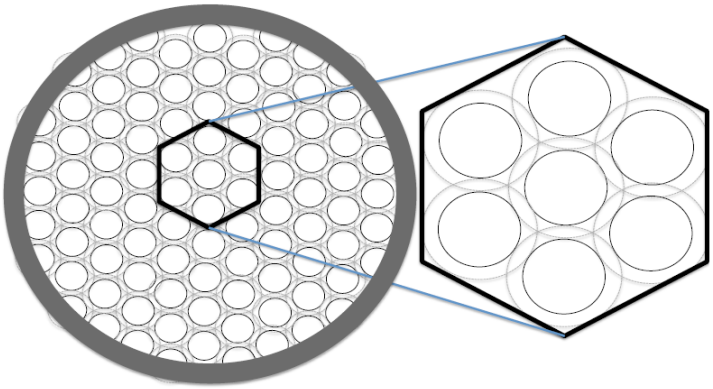


Figure 1.1: The configuration of astrobots on a focal plane (Reprinted with permission from [2])

1.2 Mechanical Characterization of the positioners

The sensor responsible for the detection of a specific type of electromagnetic waves is placed on the tip of a fiber. The fiber is connected to a spectrograph and is actuated an astrobot in the focal plane (see fig 1.2).

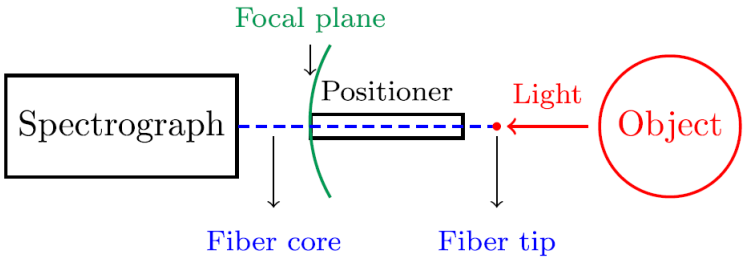


Figure 1.2: The schematic of a telescope equipped with an astrobot (Reprinted with permission from [1]).

Each astrobot is a two link planar manipulator whose end-effector has to reach a specified point at which the fiber shall observe an astronomical object.
The forward kinematics corresponding to the workspace of each astrobot is described as:

$$\mathbf{q^i} = \mathbf{q_b^i} + \begin{bmatrix} cos(\theta_i) & cos(\theta_i + \phi_i) \\ sin(\theta_i) & sin(\theta_i + \phi_i) \end{bmatrix} \cdot \mathbf{l}, \tag{1.1}$$

where $\mathbf{q^i} = [x^i,\ y^i]^T$ denotes the location of the i-th astrobot's end effector with respect to a universal reference frame attached to the focal plane of a given telescope, $\mathbf{q_b^i} = [x_b^i,\ y_b^i]^T$ denotes the base coordinate of the astrobot, $\mathbf{l} = [l_1,\ l_2]^T$ represents the lengths of the rotational links while the angular positions of the i-th astrobot are denoted by θ_i and ϕ_i (See Fig 1.3).

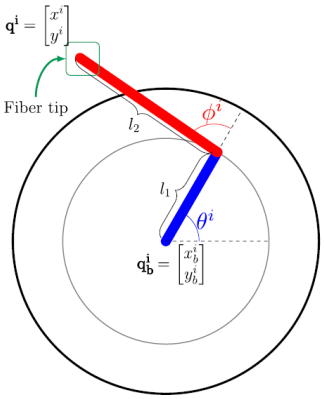


Figure 1.3: Schematic design of an astrobot (Reprinted with permission from [1])

The target position of the i-th positioner is indicated by $\mathbf{q_T^i} = [x_T^i,\ y_T^i]^T$. We can assert that the astrobot has reached its target position if its equilibrium position coincides with its target position after the coordination process, which is the result of the application of a decentralized navigation function to each astrobot [1].

1.3 Decentralized Navigation Function

In a system composed of N astrobots the objective is to coordinate each astrobot to its target while avoiding collisions with the other astrobots during their motion. The use of a centralized control algorithm would be infeasible in this framework, because of the huge number of astrobots present in the focal plane of a telescope, and the huge number of constraints to take into account for the motion of an astrobot.
The solution to the coordination problem exploits the definition of an artificial potential field with some specific characteristic which make it a navigation function. The navigation function maps the motion of each astrobot within its workspace in an analytical way. Thanks to navigation functions, an astrobot is attracted by its target position and it is repulsed by its neighbour astrobots. The main drawback of this approach is that many spurious local minima are present that affect in a drastic way the convergence of the astrobot to its target position.
A navigation function Ψ_i that regulates the motion of the i-th astrobot is composed of two terms: an attractive term and a repulsive term. The attractive term is defined as the square distance between the end effector of astrobot i and its target. The repulsive term has the objective of avoiding collisions between the astrobot and its neighbors, this term is activated when two astrobots are closer than a distance D, otherwise this term vanishes. In this way, D defines the radius of collision avoidance

region, while the parameter d represents the radius of the safety region.

$$\Psi_i = \lambda_1 ||\mathbf{q^i} - \mathbf{q^i_T}||^2 + \lambda_2 \sum_{j \neq i} \min \left(0, \frac{||\mathbf{q^i} - \mathbf{q^j}||^2 - D^2}{||\mathbf{q^i} - \mathbf{q^j}||^2 - d^2}\right) \tag{1.2}$$

Two parameters λ_1 and λ_2 are weights that regulate the importance of the two terms. They have to respect the following condition in order to ensure that the function Ψ is a navigation function:

$$\frac{\lambda_1}{\lambda_2} < \frac{1}{R} \frac{D}{D^2 - d^2} \ . \tag{1.3}$$

The control low that regulates the motion of the two arms of an astrobot is:

$$u_i = -\nabla \Psi(q_i) \ . \tag{1.4}$$

It follows that at every step an astrobot will move its respective fiber according to a gradient descent method. The angular velocities of the two motors responsible for the motion of the two arms can be obtained using the chain derivatives rule and the forward kinematics given by equation 1.1.

$$\begin{bmatrix} \omega_{i1} \\ \omega_{i2} \end{bmatrix} = u_i = - \begin{bmatrix} \frac{\partial \Psi_i}{\partial x_i} \frac{\partial x_i}{\partial \theta_i} + \frac{\partial \Psi_i}{\partial y_i} \frac{\partial y_i}{\partial \theta_i} \\ \frac{\partial \Psi_i}{\partial x_i} \frac{\partial x_i}{\partial \phi_i} + \frac{\partial \Psi_i}{\partial y_i} \frac{\partial y_i}{\partial \phi_i} \end{bmatrix} \ , \tag{1.5}$$

where ω_{i1} and ω_{i2} are the angular velocities of the first and the second motor of the i-th astrobot respecitvely.
The complexity of the algorithm based on the decentralized navigation function is $O(N)$ [3].

1.4 Prediction objectives

The use of a DNF for the coordination of each astrobot of a focal plane involves the major drawback of dealing with the presence of many local minima, which make the astrobot's arms to stumble into deadlocks. Thus, in the case of focal planes with a large number of astrobots it is not feasible to obtain a complete coordination (i.e. the convergence the of every astrobot to its target position). Although the embedding of a cooperative attractive term in equation 1.2 ensures a complete coordination of the astrobots in the focal plane [1], such a solution turns out to be temporally expensive in view of a successive reassignment of the targets of the astrobots. As a matter of fact, the available time to coordinate one astrobot to its target is limited. Indeed the observations may vary within few minutes, depending on the Earth's rotation and the positions of the particular celestial objects under study.
Therefore, the presence of astrobots which are not able to reach their targets position after the coordination process has to be taken into account. Unfortunately, the partial coordination may lead to small convergence rates which cause the lack of enough data for spectroscopic surveys. It is therefore of fundamental importance to have information as accurate as possible about which astrobots are able

to converge to their target before the start of a coordination process. In this way, if the prediction entails a low percentage of convergence, it will be possible to avoid the coordination process and go directly to the analysis of the next configuration. The overall process is depicted in figure 1.4.

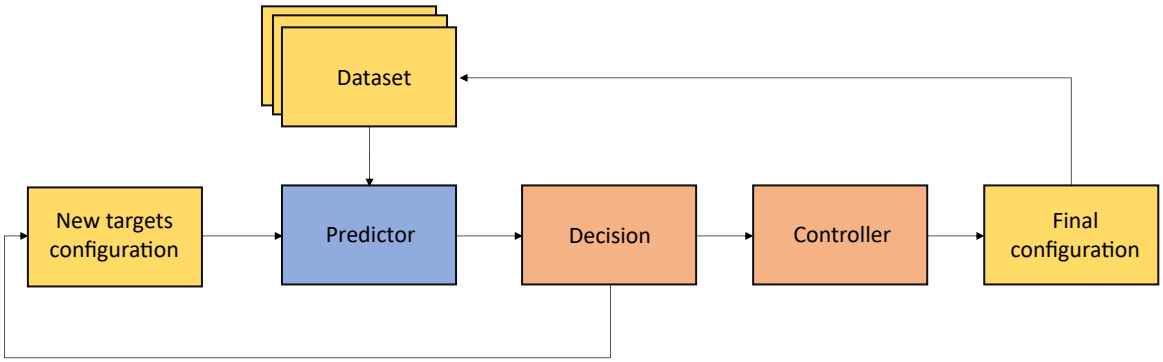


Figure 1.4: The schematic of the convergence prediction problem and its solution

Once a final configuration is reached, it will be added to a dataset used by a predictor.

Chapter 2

Parity-based prediction algorithms

The parity of an astrobot is the direction of motion of the second arm of the manipulator, which could be clockwise or counterclockwise.
In this first part of the work, the parity of the astrobot is not taken into account. Indeed in every simulated configuration of the focal plane the parity information of the astrobots is always the same, it does not change among the data. For this reason we don't have the information on the parity as a feature inside our data.

2.1 KNN-based algorithm

The KNN (K-nearest neighbors) algorithm is one of the simplest classification algorithms in machine learning. It falls in the category of instance-based learning algorithms (or lazy learning algorithms) in which the algorithm is not able to generate a global model of the data but rather it approximates the function only locally. For this reason, the KNN algorithm is very sensitive to the local structure of the data. This feature is particularly suitable for our classification problem, since the convergence of an astrobot to its target position is heavily dependent on local characteristic of the target positions of each astrobot.
To be more precise, the developed algorithm is a weighted KNN algorithm [4]. The weights are introduced in order to manage the problem of the imbalanced classes. Indeed one of the main drawbacks of the KNN is that it does not work well with imbalanced data classes. In our problem the number of times that an astrobot is able to reach its target is in general much greater than the number of times in which an astrobot stumbles into a deadlock. As a result, in our training dataset we have a prevalence of astrobots which converge to their targets positions. If weights are not used, most of the time the algorithm will predict a 1 for a specific astrobot in the configuration, since in general it is more likely that the astrobot will converge to its target. For this reason the 0s in the training configurations shall have a larger weight.

There are many parameters that need to be tuned in order to achieve a good result in terms of accuracy. Furthermore, in order to get reliable results in terms of accuracy, it is necessary to perform a cross validation, so that it is possible to get a good estimation of the efficiency of the predictor. The method adopted for the cross validation is the Monte Carlo cross validation (more details about how

to chose the number of iterations will be provided in sec. 2.1.8 and 4.4.1).

Before starting with the description of the algorithms it is necessary to introduce some formal definitions of variables and functions that will be used in the continuation.

Definition 1. An astrobot $\boldsymbol{\pi}$ is denoted as a column vector whose length equals the number of the parameters describing it, which are the coordinates of the position of its target:

$$\boldsymbol{\pi} = \begin{bmatrix} x_{tar} \\ y_{tar} \end{bmatrix} \tag{2.1}$$

Remark. In the definition above, x_{tar} and y_{tar} indicate respectively the x and y position of the target of the astrobot in the reference frame of the focal plane. We are not taking into account the position of the centre of the astrobot since it is a fixed parameter. In this framework we denote each astrobot by using only the x and y coordinates of its target.

Definition 2. A configuration $\mathbf{C}$ is a matrix of dimension $2 \times n$ that includes all the information about the targets of the astrobots in the focal plane, where 2 are the parameters that characterize the configuration and n is the number of astrobots in the focal plane.

$$\mathbf{C} = \begin{bmatrix} x_{tar_1} & x_{tar_2} & \cdots & x_{tar_n} \\ y_{tar_1} & y_{tar_2} & \cdots & y_{tar_n} \end{bmatrix} \tag{2.2}$$

Remark. In the above definition x_{tar} and y_{tar} are already defined in def. 1.

Definition 3. The ground truth associated to a configuration is a row vector $\mathbf{t}$ of length equal to the number of astrobots n, and whose elements are either 1 or 0, indicating respectively whether each astrobots has reached its target position or not after a coordination process.

Remark. The ground truth is an *a posteriori* information that can be obtained after the software simulation with the corresponding configuration.

2.1.1 Creation of the Datasets

The full dataset D has been generated by keeping the position of the center of the astrobots fixed in the focal plane, and by varying in a random way the position of the target in the patrol zone of each astrobots. Every astrobot starts its coordination process in a singular position with the angle $\theta = 0°$ and the angle $\phi = 180$ °. A further attention has been taken so that the generated targets were not too close, as this case would certainly compromise the convergence of one of the astrobots involved. This minimum distance is equal to 4 millimiters, that is the safety distance used in the definition of the decentralized navigation function algorithm.
The dataset must be as representative as possible for the prediction result to be reliable. This means

that the more configurations are simulated (and therefore more ground truth associated to a specific configuration are generated), the more the dataset represents in a detailed way the system under consideration. In fact the physical system of the focal plane is very complex, for example, just moving the target of a single astrobot a few tenths of a millimeter can result in a deadlock situation for one or more astrobots. Therefore it is necessary to have as many simulated configurations as possible, so that it is less hard for the algorithm to find the configurations "most similar" to the one under test.
Once the dataset D has been created, it is necessary to split it in training dataset D_{train} and testing dataset D_{test}. The division is made according to a specified ratio d that indicate the number of testing configurations over the total number of configuration in the dataset D. The testing configurations are picked in a random way among the full dataset D and obviously they are then not present in the training dataset D_{train}, so that $D_{train} = D \setminus D_{test}$. The value of d depends on how large the dataset D is. The more is large the the dataset D, the more we can use a good portion of it as testing set. In any case it is chosen so that D_{test} is much smaller than D_{train}, for example having a testing dataset D_{test} that is not larger than 10 % of the dataset D is a good choice.

2.1.2 Generation of the vector of weights

As already pointed out, the KNN algorithm is very sensitive to imbalanced data classes. When a configuration with a relevant number of astrobots in the focal plane is simulated, the rate of convergence of the astrobots to their target position oscillates between 65% and 95%, with the result that we have more 1s in the ground truth vector $\mathbf{t}$, and then much more 1s considering all the configurations in the training dataset D_{train}.
In general there are many ways to handle this problem in machine learning. One of them is to performing an oversampling for the minority class (the 0s in our case) or an undersampling in the majority class (the 1s) [5]. However this approach is infeasible in our case. In fact in order to perform an oversampling for the minority class, we should look for configurations with an associated ground truth vector that has more 0s than 1s (and for focal planes with a large number of astrobots these configurations don't exist or they are very rare). If we are able to find these configurations, the next step is to generate a new group of targets that is very close to the targets of the configurations that we have found. This procedure however takes a very long time, and has the risk of generating a lot of new 1s, together with the 0s, effectively canceling the purpose of oversampling. On the other hand, if we want to perform an undersampling, we should remove all the configurations with an associated ground truth vector that has more 1s than 0s (the great majority of the configurations). But the consequence is to lose an important wealth of information, which is in any case essential for the representativeness of the dataset.
A possible solution to the problem of imbalanced data classes is to use a vector of weights which has the aim of increasing the weights of the 0s in the ground truth vector of a specific configuration. The strategy adopted has many similarities with the use of the class confidence weights [4], but in our case the weight is not applied to the data sample (the configuration) but to the single astrobot.
Suppose our training dataset D_{train} has N configurations $\mathbf{C_j}$ where $j \in \{1,2,\ldots,N\}$, with consequently a number N of ground truth vector $\mathbf{t_j}$ associated to them. Let's indicate with n the number of astrobot in the focal plane, and let's call $\mathbf{u}$ a vector of length n whose elements represent the total number of times that an astrobot reach its target position for the configurations of the training dataset, that is:

$$\mathbf{u} = \sum_{j=1}^{N} \mathbf{t_j} \tag{2.3}$$

And let's call $\mathbf{v}$ the vector of length n whose elements represent the total number of times that an astrobots does not reach its target position for the configurations in the training set, that is:

$$\mathbf{v} = N \cdot \mathbf{1} - \mathbf{u} \tag{2.4}$$

Where $\mathbf{1}$ is a vector of 1s of length n and the simbol $\cdot$ indicates a normal scalar multiplication.
Let's now define the operator Ξ as follow:

Definition 4. The operator Ξ takes as input two vectors $\mathbf{u}$ and $\mathbf{v}$ of the same length and gives as output a vector $\mathbf{w}$ of the same length of $\mathbf{u}$ and $\mathbf{v}$. The output is computed as follow:

$$\Xi(\mathbf{u}, \mathbf{v}) = \mathbf{w} \text{ such that } w_i = \begin{cases} u_i / v_i \text{ if } v_i \neq 0 \\ u_i \text{ if } v_i = 0 \end{cases} \tag{2.5}$$

Where w_i, u_i and v_i are the i-th elements of the vectors $\mathbf{w}$, $\mathbf{u}$ and $\mathbf{v}$ respectively.

Then our vector of weights is the result of the application of the operator Ξ to the vectors $\mathbf{u}$ and $\mathbf{v}$.

$$\mathbf{w} = \Xi(\mathbf{u}, \mathbf{v}) \tag{2.6}$$

Remark. The operator Ξ is basically the Hadamard division [6] (or element-wise division) with an additional condition to ensure that there are not divisions by zero.

The elements w_i of the obtained vector represent the weights which have to be applied to the 0s of a specific astrobot in the configuration. The reason why it is necessary to assign a different weight to each astrobot lies in the fact that the astrobots which are in a full neighborhood configuration (See Fig 2.1) in general don't reach their target position as much as the astrobots which are not in a full neighborhood configuration. For this reason, the 0s of an astrobot in a full neighborhood configuration shall have a smaller weight, since the probability of not reaching the target for these astrobots is greater compared to the astrobots which are not in a full neighborhood configuration.

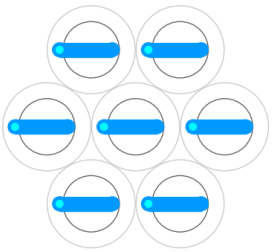


Figure 2.1: Honeycomb configuration for 7 astrobots - The central astrobot is in a "Full neighborhood" configuration.

With the defintion of the vector **w** we have managed the problem of imbalanced data classes. However in our problem the two classes have not the same importance. Indeed we are more interested in a correct prediction of the 1s from a practical point of view. It would be therefore useful to tune the weights w_i of the vector **w** according to our prediction goal. For this reason two corrector coefficients α and β are introduced. These two coefficients will be further characterized in sec. 4.1, their role consists in modifying the weights of the 0s depending on which class we want to give more importance.

2.1.3 Definition of the distance metric

In the following, we will indicate with **T** a test configuration (a configuration that belongs to the set D_{test}) and with **C** a training configuration (a configuration that belongs to D_{train}). The KNN algorithm classifies a new test configuration **T** on the basis of the ground truth vectors **t** associated to configurations **C** in the training set which are "closest" to the test configuration. It is therefore crucial to define a distance metric between two configurations, and on the basis of this metric evaluate then the proximity between two configurations.

Metric 1 - Sum of the euclidean distance between targets
Given a test configuration **T** and a train configuration **C** we define metric d_1 as:

$$d_1(\mathbf{T}, \mathbf{C}) = \sum_{k=1}^{n} \sqrt{(x_{T\ tar_k} - x_{C\ tar_k})^2 + (y_{T\ tar_k} - y_{C\ tar_k})^2} \tag{2.7}$$

Where the notation $x_{T\ tar_k}$ indicates the x position of the target of the k-th astrobot of the test configuration **T**, while $y_{T\ tar_k}$ indicates the y position of the target of the k-th astrobot of the test configuration **T**. The same holds for the train configuration **C**.

2.1.4 K-nearest neighbours

Once the metrics have been defined, it is possible to describe how the K closest configurations are chosen. Before starting, it is necessary to underline the role of the hyperparameter K in the algorithm.

This parameter has to be chosen in a smart way, depending on the size of the training dataset and on the complexity of the system in analysis. It must not be too small, otherwise there is some risk to overfit the test configuration. On the other hand K should not be too large, otherwise there is the risk to take in consideration also train configurations that are sufficiently different from the testing one, and this would lead to have more astrobots whose prediction is wrong.
The K nearest neighbours are selected according to the distance d_1, as decribed below:

Given a test configuration $\mathbf{T}$ and the training dataset D_{train} of cardinality N we use the distance d_1 as metric for the distance between the test configuration and the training configurations $\mathbf{C_j}$, with $j \in \{1,2,\ldots,N\}$:

$$\forall\ \mathbf{C_j} \in D_{train},\ d_j = d_1(\mathbf{T}, \mathbf{C_j}) \tag{2.8}$$

Once we have all the distances d_j with $j \in \{1,2...,N\}$, we sort them in ascending order and take the first K configurations $\mathbf{C_j^\star}$ corresponding to the first K smallest distances. The notation $\mathbf{C_j^\star}$ indicates the the closest K train configurations to a given test configuration $\mathbf{T}$.

2.1.5 Probability computation

Once the K closest train configurations $\mathbf{C_j^\star}$ are obtained, it is possible to compute the probability vector $\tilde{\mathbf{p}}$ associated to a test configuration $\mathbf{T}$. In order to determine this vector we use the ground truth vectors $\mathbf{t_j^\star}$ associated to the closest train configurations and the vector of weights $\mathbf{w}$.
We recall that the elements of the vector $\mathbf{w}$ represent the weights that shall be given to the 0s of a specific astrobot. For example, if the i-th element of $\mathbf{w}$ is equal to 4, then the weight that should be given to the 0 of the i-th astrobot is 4. It follows that the vector of weights $\mathbf{w}$ has the role of increasing the weights of the 0s in each ground truth vector $\mathbf{t_j^\star}$. But not all the elements of the ground truth vector are 0s. Indeed, since there are many 1s in the vectors $\mathbf{t_j^\star}$, it is then necessary to define an operator Υ which makes sure that the weights of the vector $\mathbf{w}$ are applied only to the 0s of the vector $\mathbf{t_j^\star}$. The output of this operator is a vector $\bar{\mathbf{w}}$ which represent the vector of weights for a specific ground truth $\mathbf{t}$.

Definition 5. The operator Υ takes as input the vector of weights $\mathbf{w}$ and a ground truth vector $\mathbf{t}$ and gives as output a vector $\bar{\mathbf{w}}$ of length n, that is the number of astrobots. An element $\bar{w_i}$ of the vector $\bar{\mathbf{w}}$ is set to 1 if the corresponding element t_i in the ground truth vector $\mathbf{t}$ is 1, and is set to the element w_i of the vector of weights $\mathbf{w}$ if the corresponding element t_i of the ground truth vector $\mathbf{t}$ is 0.

$$\Upsilon(\mathbf{t}, \mathbf{w}) = \bar{\mathbf{w}} \ \text{ such that } \ \bar{w_i} = \begin{cases} 1 \text{ if } t_i = 1 \\ w_i \text{ if } t_i = 0 \end{cases} \quad ,\ \ i \in \{1,2,\ldots,n\} \tag{2.9}$$

Example
Given $\mathbf{w} = [3.5\ 4\ 5\ 3]$ and $\mathbf{t} = [0\ 1\ 0\ 1]$, then $\Upsilon(\mathbf{t}, \mathbf{w}) = \bar{\mathbf{w}} = [3.5\ 1\ 5\ 1]$.

The operator $\boldsymbol{\Upsilon}$ is then applied to the ground truth vectors $\mathbf{t_j^\star}$ of the closest K train configurations $\mathbf{C_j^\star}$, obtaining the vectors of weights $\bar{\mathbf{w}}_\mathbf{j}{}^\star$ which represent the specific vector of weights for the ground truth vector $\mathbf{t_j^\star}$.

$$\bar{\mathbf{w}}_\mathbf{j}{}^\star = \Upsilon(\mathbf{t_j^\star}, \mathbf{w}) \tag{2.10}$$

Finally, the method chosen for calculating the probability vector is an element-wise [6] weighted average of the K ground truth vector $\mathbf{t_j^\star}$ using the specific vector of weights $\bar{\mathbf{w}}_\mathbf{j}{}^\star$.

$$\tilde{\mathbf{p}} = \left(\sum_{j=1}^{K} \mathbf{t_j^\star}\right) \oslash \left(\sum_{j=1}^{K} \bar{\mathbf{w}}_\mathbf{j}{}^\star\right) \tag{2.11}$$

Example
Suppose we want compute the probability $\tilde{\mathbf{p}}$ using the closest two train configurations (i.e. K = 2). In a focal plane with $n = 5$ astrobots, suppose the ground truth vectors $\mathbf{t_j^\star}$ of our closest train configurations $\mathbf{C_j}^\star$ with $j \in \{1,2\}$ are:

$$\mathbf{t_1^\star} = [1\ 0\ 1\ 0\ 1] \quad \mathbf{t_2^\star} = [1\ 0\ 1\ 1\ 1]$$

Suppose now the vector of weights $\mathbf{w}$ is equal to [3 4 3.5 3 3]. If we apply the operator Υ to the ground truth vectors $\mathbf{t_1^\star}$ and $\mathbf{t_2^\star}$, we obtain:

$$\bar{\mathbf{w}}_\mathbf{1}{}^\star = \Upsilon(\mathbf{t_1^\star}, \mathbf{w}) = [1\ 4\ 1\ 3\ 1] \quad \bar{\mathbf{w}}_\mathbf{2}{}^\star = \Upsilon(\mathbf{t_2^\star}, \mathbf{w}) = [1\ 4\ 1\ 1\ 1]$$

If we now perform the two summation in the equation 2.11 we obtain:

$$\sum_{j=1}^{2} \mathbf{t_j^\star} = [2\ 0\ 2\ 1\ 2] \quad \sum_{j=1}^{2} \bar{\mathbf{w}}_\mathbf{j}{}^\star = [2\ 8\ 2\ 4\ 2]$$

The probability $\tilde{\mathbf{p}}$ is then:

$$\tilde{\mathbf{p}} = [2\ 0\ 2\ 1\ 2] \oslash [2\ 8\ 2\ 4\ 2] = [1\ 0\ 1\ 0.25\ 1]$$

Remark. The notation $\tilde{\mathbf{p}}$ is needed to ensure consistency with the section 2.1.6. Indeed we will see that the final probability $\mathbf{p}$ is computed using the differents probabilities $\tilde{\mathbf{p_k}}$ corresponding to differents neighborhood of the focal plane, with $k \in \{1,2,\cdots,n\}$ and n which is the number of astrobots.

It is possible to use more sophisticated methods to compute the probability vector, like using a different weight for each ground truth vector on the basis of the distance from the testing configuration $\mathbf{T}$, but the risk would still be to fall into a possible overfitting. For this reason, to keep things more simple we opted to avoid giving weights also to the configurations $\mathbf{C_j^\star}$ and then to the ground truth vector $\mathbf{t_j^\star}$.

2.1.6 Local neighborhood analysis

When dealing with very crowded astrobots swarms, the metric described above is not sufficient to assess the similarity between two configurations. This is because the huge number of astrobots would make the information on the distance between the targets unreliable, since it could happen that even in the closest training configuration there are targets of many astrobots that are very far with respect to the ones in the testing configuration (see Fig. 2.2).

There are two possible solutions to this problem. The first one is to have a very huge training dataset

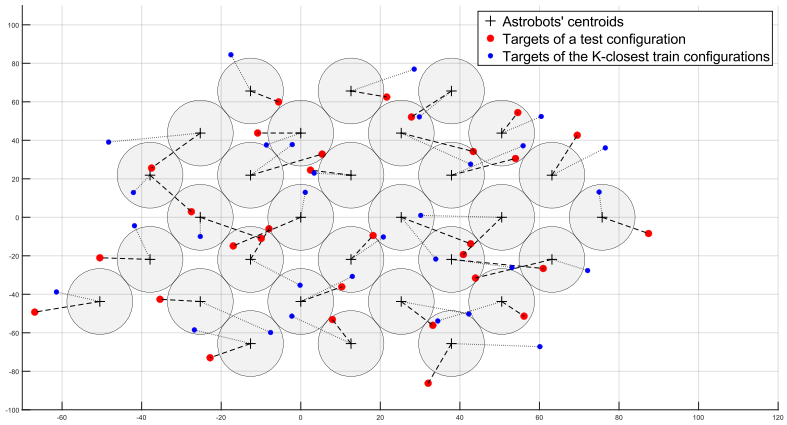
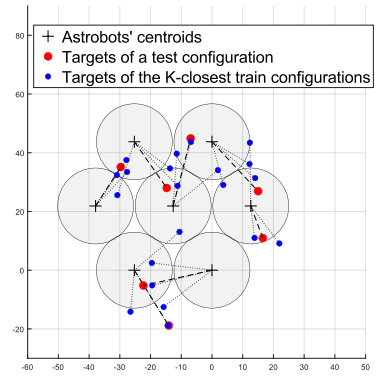
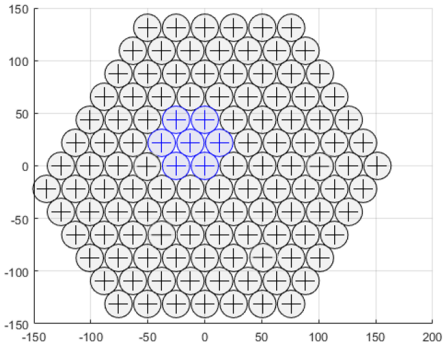


Figure 2.2: Focal plane with 30 astrobots. The red dots represent the target of the test configuration, the blue dots represent the targets of the closest train configuration according to the distance d_1. it is possible to observe that for many astrobots, the distance between the target of the test configuration and the target of the train configuration is very high.

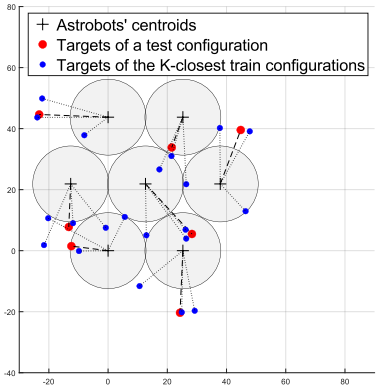
(that more or less covers all the possible situations for each combination of the position of the target of the astrobots). But this solution would be pratically infeasible since it would require an enormous amount of time and memory storage. The second solution consists of performing a local analysis on the neighbours of each astrobot. In this way the algorithm is recursively applied to a number of small configurations with a maximum number of astrobots equal to 7 (each astrobot can be surrounded by a maximum of 6 other neighbours). This kind of approach limits the possibility of having astrobots whose distance between the targets in the test configuration and the targets in the trainings configuration is high (see Fig 2.3).



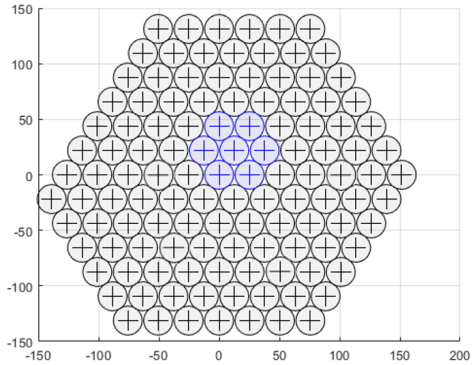
(a)



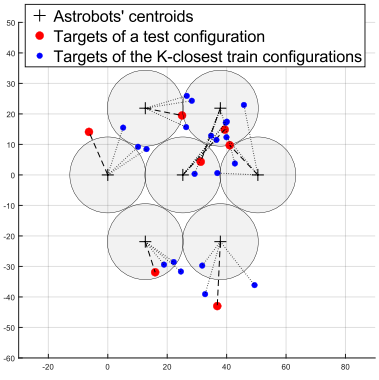
(b)



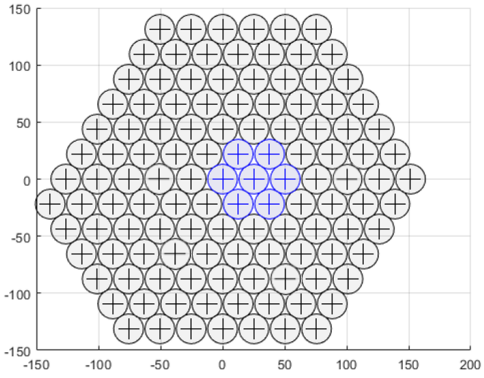
(c)



(d)



(e)



(f)

Figure 2.3: Local analysis on neighborhoods with 7 astrobots in a focal plane with 116 astrobots. Figure (a), (c) and (e) show local analysis performed on the neighborhoods using the closest three train configuration, while Figure (b), (d) and (f) show the position of the neighborhoods inside the focal plane. In each case it is possible to notice that the blue dots (targets of the train configurations) are now nearby the red dots (targets of the test configuration).

Definition 6. A neighborhood $\boldsymbol{\nu(\pi)}$ of an astrobot $\boldsymbol{\pi}$ is a subset of a configuration $\mathbf{C}$:

$$\boldsymbol{\nu(\pi)} \subseteq \mathbf{C} \tag{2.12}$$

In particular, $\boldsymbol{\nu(\pi)}$ is a matrix of dimension $2 \times r$, where the two rows are for the parameters that characterize the targets of astrobots of the neighborhood and r are the number of the astrobots that are in the neighborhood, including the central astrobot $\boldsymbol{\pi}$.

Remark. The number of columns of $\mathbf{C}$ is n, which is the total number of astrobots in the focal plane. From the definition it follows that $r \leq n$. The dimension r varies between 1 and 7. It is 1 when the astrobot is isolated (no neighbours surround it) and it is 7 when the astrobot is in a full neighborhood configuration.

If we have n astrobots in the focal plane, we should perform n local analyses, since for each astrobot we have only one neighborhood in which the astrobot itself is in the central position.
In order to perform this kind of analysis, it is necessary to introduce an auxiliary vector $\boldsymbol{\eta}$.

Definition 7. The vector $\boldsymbol{\eta}$ has a length equal to the number of astrobots n. Its elements are integers which represent the number of times a specfic astrobot appears in every possible neighborhood $\boldsymbol{\nu(\pi)}$ it can belong to when performing a local analysis.

Remark. The i-th element of the vector represents the number of times the i-th astrobot appears as a neighbour, including the case where it is the central astrobot of the neighborhood $\boldsymbol{\nu(\pi)}$. This means that the elements of the vector $\boldsymbol{\eta}$ are integer numbers between 1 and 7.

To compute the final probability $\mathbf{p}$ for the entire focal plane, it is necessary to introduce the local probability $\tilde{\mathbf{p}}_\mathbf{k}$ for a given neighborhood $\boldsymbol{\nu(\pi_k)}$.

Definition 8. The local probability vector $\tilde{\mathbf{p}}_\mathbf{k}$ of a given neighborhood $\boldsymbol{\nu(\pi_k)}$ is a vector of length equal to the number of astrobots n, whose elements $\tilde{p}_{k\ i}$, with $i \in \{1,2,\cdots,n\}$, are defined as follows:

$$\tilde{p}_{k\ i} = \begin{cases} 0 \text{ if } \boldsymbol{\pi_i} \notin \boldsymbol{\nu(\pi_k)} \\ \tilde{p}_i \text{ if } \boldsymbol{\pi_i} \in \boldsymbol{\nu(\pi_k)} \end{cases} \quad , \quad i \in \{1,2,\ldots,n\} \tag{2.13}$$

Where $\tilde{p}_i$ is an element of the vector $\tilde{\mathbf{p}}$ defined using eq. 2.11, and computed using the neighborhood $\boldsymbol{\nu(\pi_k)}$ as configuration.

Remark. The result of the neighborhood analysis of the k-th astrobot $\boldsymbol{\pi_k}$ is a probability vector $\tilde{\boldsymbol{p}}_\boldsymbol{k}$ of length n. The i-th element of this probability vector is 0 if the i-th element corresponds to an astrobot $\boldsymbol{\pi_i}$ which is not in the neighborhood of the k-th astrobot $\boldsymbol{\nu(\pi_k)}$. While if the element correspond to an astrobot which is in the neighborhood $\boldsymbol{\nu(\pi_k)}$, then this element correspond to the element $\tilde{p}_i$ of the probability vector $\tilde{\mathbf{p}}$ computed as described above (section 2.1.5), but using the neighborhood $\boldsymbol{\nu(\pi_k)}$ as testing configuration.

Let's make an example: suppose we have a configuration with n=10 astrobots. We start the local analysis from the astrobot $\boldsymbol{\pi_1}$. Suppose the neighbours of this astrobot are $\boldsymbol{\pi_2}$, $\boldsymbol{\pi_3}$ and $\boldsymbol{\pi_5}$. Then the neighborhood $\boldsymbol{\nu(\pi_1)}$ is a matrix of 4 columns. At this point the neighborhood $\boldsymbol{\nu(\pi_1)}$ becomes the test configuration for the procedures described in sections 2.1.4 and 2.1.5. The probability vector $\tilde{\mathbf{p}}_\mathbf{1}$ that we obtain is a vector of length n which has all zeros except for the elements corresponding to the indices i=1,2,3 and 5, since these are the indeces corresponding to the astrobots in the neighborhood $\boldsymbol{\nu(\pi_1)}$. The elements corresponding to the indices i=1,2,3 and 5 represent the probabilities computed in the section 2.1.5.
This procedure is applied to each astrobot of the configuration. The final probability $\mathbf{p}$ for the test configuration $\mathbf{T}$ is then computed as follow:

$$\mathbf{p} = \left(\sum_{k=1}^{n} \tilde{\mathbf{p}}_\mathbf{k} \right) \oslash \boldsymbol{\eta} \tag{2.14}$$

Example
Let's suppose for simplicity that we have only 5 astrobots in the focal plane and they are arranged in two groups of three and two astrobots, as shown in Figure 2.4:

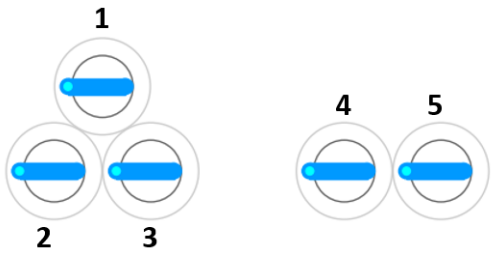


Figure 2.4: Astrobots in the focal plane arranged in two groups of three and two astrobots

The vector $\boldsymbol{\eta}$ in this case has five elements and it is $\boldsymbol{\eta}$ = [3, 3, 3, 2, 2]. Indeed the astrobot $\boldsymbol{\pi_1}$ appears as neighbour in three neighborhoods which are $\boldsymbol{\nu(\pi_1)}$, $\boldsymbol{\nu(\pi_2)}$ and $\boldsymbol{\nu(\pi_3)}$. The same for the astrobots $\boldsymbol{\pi_2}$ and $\boldsymbol{\pi_3}$. While the astrobot $\boldsymbol{\pi_4}$ appears as neighbour in two neighborhoods which are $\boldsymbol{\nu(\pi_4)}$ and $\boldsymbol{\nu(\pi_5)}$, and the same for the astrobot $\boldsymbol{\pi_5}$.
Let's say now that the vectors $\tilde{\mathbf{p}}_\mathbf{k}$ with $k \in \{1,\ 2,\ 3,\ 4,\ 5\}$ and computed according to the procedures described in sections 2.1.4 and 2.1.5 are:

$$\tilde{\mathbf{p}}_\mathbf{1} = [1\ 0.8\ 0.7\ 0\ 0]; \quad \tilde{\mathbf{p}}_\mathbf{2} = [1\ 0.6\ 0.8\ 0\ 0]; \quad \tilde{\mathbf{p}}_\mathbf{3} = [1\ 0.9\ 0.9\ 0\ 0];$$

$$\tilde{\mathbf{p}}_\mathbf{4} = [0\ 0\ 0\ 1\ 0.8]; \quad \tilde{\mathbf{p}}_\mathbf{5} = [0\ 0\ 0\ 0.8\ 0.9]$$

The sum of these vectors is:

$$\sum_{k=1}^{5} \tilde{\mathbf{p}}_{\mathbf{k}} = [3\ 2.3\ 2.4\ 1.8\ 1.7]$$

The final probability vector $\mathbf{p}$ is then:

$$\mathbf{p} = \left(\sum_{k=1}^{5} \tilde{\mathbf{p}}_{\mathbf{k}} \right) \oslash \boldsymbol{\eta} = [3\ 2.3\ 2.4\ 1.8\ 1.7] \oslash [3\ 3\ 3\ 2\ 2] = [1\ 0.766\ 0.8\ 0.9\ 0.85]$$

Remark. The numerical values used in this example are not taken from a real simulation of the algorithm. They have the sole purpose of showing how local analysis is done.

2.1.7 Output of the prediction

Before starting with the description of how the output vector is computed, it is necessary to introduce the operator $\boldsymbol{\Theta}$.

Definition 9. The operator $\boldsymbol{\Theta}$ takes as input the vector of probability $\mathbf{p}$ of length n, which is the number of astrobots, and a scalar q and gives as output a vector $\mathbf{y}$ whose elements y_i are 0s and 1s, according to the following rule:

$$\Theta(\mathbf{p}, q) = \mathbf{y} \ \text{ such that } \ y_i = \begin{cases} 1 \text{ if } p_i > q \\ 0 \text{ if } p_i \leq q \end{cases} \quad , \ \ i \in \{1,2,\ldots,n\} \tag{2.15}$$

Where p_i denotes the i-th element of the vector $\mathbf{p}$.

With the probability vector $\mathbf{p}$ it is possible to compute the output of the prediction $\mathbf{y}$ if we are given a treshold q.

$$\mathbf{y} = \Theta(\mathbf{p}, q) \tag{2.16}$$

Example
Given $\mathbf{p} = [0.89\ 0.4\ 0.67]$ and $q = 0.5$, then $\Theta(\mathbf{p}, q) = \mathbf{y} = [1\ 0\ 1]$.

The treshold q is set to 0.5. The use of the weights for the 0s makes prior information of the class lebels symmetric. Indeed the use of weights balances the frequency of the 0s with the frequency of the 1s when the computation of the probability is performed, and then it is reasonable to use a treshold that is symmetric.

2.1.8 Monte Carlo cross validation

In order to assess whether the results of the prediction are reliable without the risk of falling into overfitting, it is necessary to perform a cross validation process. The method chosen for this algorithm is a Monte Carlo cross validation [7]. There are several reasons behind choosing this form of cross validation for this algorithm. The main reason behind this choice is the computational saving with respect to a k-Fold cross validation. Indeed with a MC cross validation the proportion of the training/testing split does not depend on the number of iterations (that is the number of partitions in the k-Fold cross validation). In this way we can perform a number of iterations which is not linked to the dimension of the testing set with respect to the dimension of the training set. The drawback of this method is that some configurations may never be selected as testing configurations, whereas others may be selected more than once. For this reason it is necessary to put a particular attention to the number k of iterations of the process. The choice of the number k depends on how large is the testing dataset compared to the training dataset. The more D_{test} is small, the more k has to be large.

2.1.9 Complete algorithm

The algoritm takes as input dataset of configuration D and gives as output the accuracy of the predictor. The procedure is summarized in the following steps:

1) Split the starting dataset D into training dataset D_{train} and testing dataset D_{test}, according to a specified ratio $d = |D_{test}|/(|D_{train}| + |D_{test}|)$, where the notation $|\ .\ |$ indicates the cardinality of the dataset (i.e. the number of configurations in the dataset).

2) Generate the vector of weights $\mathbf{w}$. This vector is a row vector of length equal to the number of astrobots n.

3) Take a configuration $\mathbf{T}$ in D_{test}.

4) According to a defined metric d, for every configuration $\mathbf{C_j}$ in D_{train} compute the distance $d_j = d(\mathbf{T}, \mathbf{C_j})$ between the test configuration $\mathbf{T}$ and the training configuration $\mathbf{C_j}$.

5) Take K configurations in the D_{train} corresponding to the smallest K distances d_j between the test configuration $\mathbf{T}$ and all the training configurations $\mathbf{C_j}$.

6) Take the ground truth vectors $\mathbf{t_j^\star}$ associated with the K closest training configuration $\mathbf{C_j^\star}$ and compute the probability vector $\mathbf{p}$ using the local analysis.

7) Compute the output of the prediction $\mathbf{y} = \Theta(\mathbf{p}, q)$.

8) Go back to point 3 and repeat the procedure for all the configuration $\mathbf{T}$ in D_{test}

9) Compute the accuracy of the overall prediction, given the output vectors $\mathbf{y}$ and the ground truth $\mathbf{t}$ associated to the testing configurations $\mathbf{T}$.

10) Perform the cross validation by repeating the points 1 to 9 a number k of times and then averaging the results.

2.2 SVM algorithm

The second parity-based prediction algorithm developed is a support vector machine algorithm. It is a supervised learning algorithm which can be used both for regression and for classification. The advantages of this algorithm lie in the fact that it is efficient with high dimensional data and it uses only a subset (support vectors) of the training data in order to build the boundary between the two classes of the problem.
Before describing the algorithm and choice of the hyperparameters it is necessary to characterize the data which are provided to the algorithm. In fact, compared to the KNN algorithm, in which the data were entire configurations of the astrobots in the focal plane, now a single sample consists of the parameters that characterize a single astrobot and its neighbors. Then the main difference from KNN is that in this case the algorithm is applied to a single astrobot and not to the entire focal plane. On one hand, the classification problem is similar to the local analysis performed with the KNN (indeed also in this case we are considering the neighbours of the astrobot), but the difference is that now the binary output of the classification is just for the central astrobot of the neighborhood (while in the local analysis with KNN we considered also the ground truth of the neighbors). This approach allows to carry out a local analysis starting from the definition of the data itself, since we are including the targets coordinates of an astrobot's neighbours in a single data sample.
In this framework it is necessary to change the definition 1 of the astrobot $\boldsymbol{\pi}$:

Definition 10. An astrobot $\boldsymbol{\pi}$ is denoted as a column vector whose length equals the number of the parameters describing it and its neighbours:

$$\boldsymbol{\pi} = \begin{bmatrix} x \\ y \\ x_{n\ 1} \\ y_{n\ 1} \\ \vdots \\ x_{n\ r} \\ y_{n\ r} \end{bmatrix} \tag{2.17}$$

Where x, y denote the x and y position respectively of the target of the central astrobot, r indicates the number of neighbours of the astrobot. The parameters $x_{n\ i}$ and $y_{n\ i}$ with $i \in \{1, \dots, r\}$ denote respectively the x and y position of the target of the i-th neighbour of the astrobot.

In the case of parity-based prediction, the vector does not include the information on the parity, since we know that it is fixed for each astrobot.

A data sample in this case consists of a single astrobot, and then the ground truth is no more a vector but rather a single element t which can be either 0 or 1. In this way, assuming we have a dataset of N simulated configuration, we will have a dataset of N samples for each astrobot. Let's make an example:

Example
Suppose our focal plane has 100 astrobots and we have 2000 simulated configurations of the astrobots in the focal plane. For each astrobot in the focal plane then we have a dataset of 2000 samples and 2000 ground truth element t, where a sample is a vector as described in equation 2.17.

From this moment the term "vector" is used to indicate a single data in the dataset while with the term "feature" is used to indicate an entry of the vector $\boldsymbol{\pi}$ defined using eq. 2.17.

2.2.1 Data Preprocessing

From def. 10 it is possible to observe that the features of a single data sample are the coordinates of the targets of the astrobots in a given neighborhood. The only preprocessing step applied to the data before they are used in the algorithm is a feature scaling. The scaling is needed mainly because of two reasons. The first one is that because the target's coordinates of the different neighbours have not all the same range of variation among the data. The second one is that the coordinate values depend on the particular position of the neighborhood in the focal plane. Furthermore the SVM algorithms are not scale invariant. This means that the choice of not scaling the data would have direct consequences on the choice of the hyperparameters. In the case of the KNN a feature scaling would have changed nothing in terms of results, since we used the euclidean distance as metric to evaluate the similarity between configurations, and the distance is a relative measure. This means that if two configurations are similar without feature scaling, they remains similar even with the feature scaling. Also the SVM algorithm uses the euclidean distance, but in a different manner. Indeed we will see that the euclidean distance is used inside the Gaussian kernel which maps the distance between the vectors in the an higher dimensional space.
In particular the method adopted is a min-max normalization [8]. This method applies a linear transformation to the data's features in order to scale them in a specific range. The range for our case is the interval [-1,1]. Several simulations have confirmed that this range ensures good performance of the predictor. Furthermore, the use of this range allows to have all the features with zero mean. If we indicate with x a generic feature of the data, the formula applied to the data's features is:

$$x' = \frac{2 \cdot (x - \min(x))}{(\max(x) - \min(x))} - 1 \tag{2.18}$$

Where x' indicates the new scaled feature, $\min(x)$ is the minimum valued assumed by the feature among all the data, $\max(x)$ is the maximum value addumed by the feature among all the data.
The eq. 2.18 is applied to all the data, including the ones in the test set. It is worth noting that the transformation preserves the information of the relative position between the targets of the neighbohrood, since it is just a linear operation which scales the feature in a given range. There are other types of linear transformation, like the Z-score normalization, but several tests have shown that min-max normalization is the one that guarantees better performance. Furthermore, Z-score normalization results particularly suitable when the data follow a gaussian distribution, but this is not the case since the coordinates of the targets are randomly generated using a standard uniform distribution.

2.2.2 SVM predictor

The SVM algorithm finds a boundary between the vectors of the two classes by solving the following constraint-based optimization problem [9]:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C_0 \sum_{i=1}^{N_0} \xi_{0\ i} + C_1 \sum_{i=1}^{N_1} \xi_{1\ i} \\ \text{subject to} \quad & y_{0\ i}(\mathbf{w}\phi(\boldsymbol{\pi}_i) + b) \geq 1 - \xi_{0\ i} \\ & y_{1\ i}(\mathbf{w}\phi(\boldsymbol{\pi}_i) + b) \geq 1 - \xi_{1\ i} \\ & \xi_{0\ i} \geq 0 \\ & \xi_{1\ i} \geq 0 \end{aligned} \tag{2.19}$$

The boundary between the two classes is also called hyperplane and it is denoted using the normal vector $\mathbf{w}$. C_0 and C_1 are the two missclassification penalties for the class of 0s and 1s respectively (see sec. 4.1), The quantities $\xi_{0\ i}$ and $\xi_{1\ i}$ represent the slack variables associated to the incorrect classification of the i-th samples for the class of 0s and 1s respectively, while $y_{0\ i}$ and $y_{1\ i}$ represent the true label (ground truth) of the i-th sample for the class of 0s and 1s respctively. The i-th data sample is denoted with $\boldsymbol{\pi_i}$, while the notation $\phi(\boldsymbol{\pi_i})$ denotes the mapping of $\boldsymbol{\pi_i}$ into an higher dimensional space. Finally, b denotes the value of the intercept term of the hyperplane. N_0 and N_1 represent the number of samples in the class of 0s and 1s respectively.
The normal vector $\mathbf{w}$ and the value of the intercept b are optimization variables of the problem. The value of b is important in the linear SVM, since if the data have not zero mean, its value would be different from zero. In our case, with a strong nonlinear problem, b may be different from 0, but its value has not a particular meaning since it is difficult to visualize an hyperplane when the dimension are greater then 3. The slack variables $\xi_{0\ i}$ and $\xi_{1\ i}$ represent the error due to an incorrect classification: they are 0 if a sample has been correctly classified, while they are greater than 0 if a sample has been missclassified.
The distance between the hyperplane and the support vectors is called margin [10]. The term $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ in eq. 2.19 represents the convex form of the inverse of the margin between the two classes. Minimize this quantity is equivalent to maximize the margin. Indeed the central idea of a support vector classifier is to maximize the margin under some classification constraints. When the classification problem is complex, it is better to introduce some slack variables ξ_i in order to relax the classification constraints and allow the missclassification of some data samples. The weight given to these missclassifications is given by the penalty C.
The optimization problem is solved thanks to the sequential minimal optimization algorithm (SMO) [11]. It is important to underline that not all the data samples are needed to build the hyperplane, but rather only a subset of them, which are called support vectors. The support vectors are the data samples which are needed to construct the boundary, and indeed they are the closest data to the boundary after this one is builded, since the boundary uses them as "support".

When the data are not linearly separable, it is necessary to apply a mathematical transformation called

Kernel trick [12]. The data $\boldsymbol{\pi_i}$ are mapped into an higher dimensional space through a function $\phi(\boldsymbol{\pi_i})$. The advantage of this projection is that if the data are not linearly separable in the original feature space (if a data has five features, its original feature space has five dimensions), they might become linearly separable in an higher dimensional space (see Fig. 2.5).

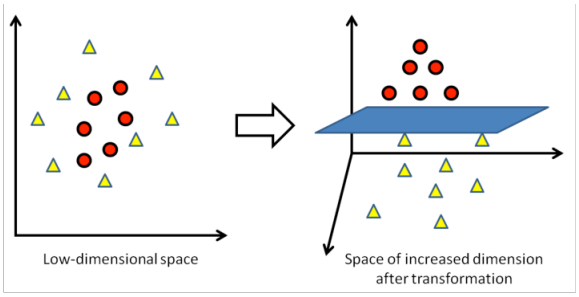


Figure 2.5: Kernel trick explicative picture (Reprinted with permission from [13])

The function that maps the distance between two data samples $\boldsymbol{\pi}$ and $\boldsymbol{\pi'}$ from the original feature space into the new higher dimensional space is called kernel. There are many types of kernel functions. The one that we use for our classification is the Gaussian kernel, which belongs to a class of kernel functions called RBF (radial basis functions) [14].

$$\kappa(\boldsymbol{\pi}, \boldsymbol{\pi'}) \equiv \phi(\boldsymbol{\pi})^T \phi(\boldsymbol{\pi'}) = \exp\left(-\frac{||\boldsymbol{\pi} - \boldsymbol{\pi'}||^2}{2\sigma^2}\right) = \exp(-\gamma||\boldsymbol{\pi} - \boldsymbol{\pi'}||^2) \tag{2.20}$$

Where κ is denotes the kernel function, $\boldsymbol{\pi}$ and $\boldsymbol{\pi'}$ are two vectors (two samples) of the dataset and σ is the kernel size. γ is another parameters which indicate the kernel size, following the relation:

$$\sigma = \frac{1}{\sqrt{2\gamma}} \tag{2.21}$$

The kernel size σ is also called bandwidth and indicates the width of the Gaussian kernel function in equation 2.20. The larger σ is, the wider the Gaussian function will be.

The linear hyperplane in the higher dimensional space becomes nonlinear in the original feature space of the data. In this way it is possible to obtain a nonlinear boundary between the data.
Once the hyperplane is found, it is possible to assign a new test vector to one of the two classes of the problem, depending on the position of the vector with respect to the hyperplane.
When there are vectors that are missclassified, the slack variables allow to manage these missclassification errors inside the optimization problem. We can tune the weight given to the missclassification term of the optimization problem by introducing a parameters called C. This parameter directly modifies the missclassification penalties C_0 and C_1 that appear in eq. 2.19. The relation between C, C_0 and C_1 is analyzed in sec. 4.1. If we increase C we will increase the missclassification penalty term and the hyperplane will have a different shape. Also in this case, as well as in the KNN algorithm, we have to consider the imbalanced data classes of the problem. In particular the solution is always to use

class weights, but for the SVM algorithm we chose to apply the weights to the majority class, which is the class of 1s (while in the KNN we applied the weights to the class of 0s). The hyperparameter which regulates the weight of the class of positives is w_1. In terms of results, there is no difference in giving a lower weight to the majority class or a bigger weight to the minority class. An idea of what is behind the class weights for SVM is provided in sec. 4.1.

The SVM model of the data is built using only the training dataset D_{train}. The performances of the algorithm are evaluated using the test dataset D_{test}.

2.2.3 k-Folds cross validation

The method chosen in this case to validate the results is a k-folds cross validation. In this case the number of iterations k depends on the dimension of the test set D_{test}. In particular, at every iteration we split the data in a sequential fashion, according to the Fig. 2.6.

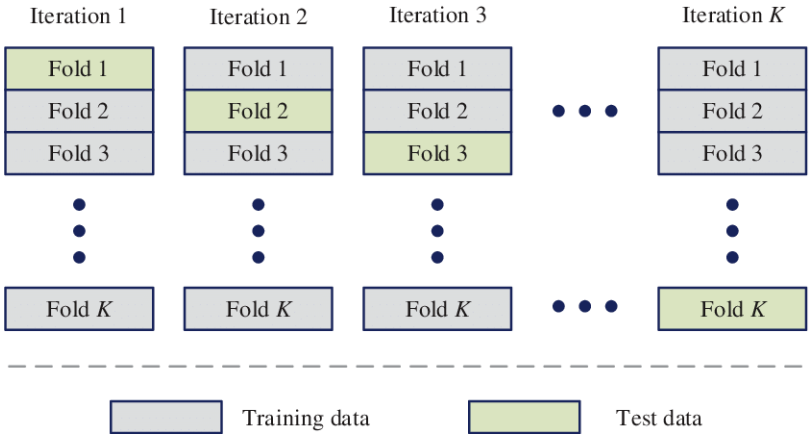


Figure 2.6: k-Folds cross validation (Reprinted with permission from [15])

For example, if our D_{test} is 10% of the overall dataset D, we should perform 10 iterations, taking every time a different portion of the full dataset as D_{test}. The performance results obtained at the end of every iteration are then averaged over the total number of iterations k in order to obtain the final results of the cross validation process.
The advantages of this method are that we use all the data as test data and train data at the end of the cross validation. Moreover there is not the risk of using more than one time the same data as test sample (as it happens for the Monte-Carlo cross validation).
The value of k depends on the ratio D_{test}/D. In our case we chose to use a testing set that is 10% of the overall dataset D. This choice is due to the fact that also in this case (as for the KNN) we want to use as many training data as possible to build the SVM model. With this ratio D_{test}/D the values of k is set to 10. If we decrease the dimension of D_{test}, we would get an higher value of k and the time required to run the cross validation would become very high. On the other hand, if we increase the dimension of D_{test}, we would use more samples as testing data, with the risk of not having enough training samples to build a representative SVM model.

2.2.4 Complete algorithm

Suppose we have a number N of simulated configurations and a number r of astrobots in the focal plane. For each astrobot $\boldsymbol{\pi_i}$ with $i \in \{1,2,\ldots,r\}$ then we have a dataset D_i composed of N data samples.
The complete SVM predictor takes as input all the dataset D_i of each astrobot and gives as output the accuracy and other performance indeces of the predictor. The procedure is summarized in the following steps:

1) Pre-process all the data by translating the targets information of every astrobot in the neighborhood near the origin of the focal plane.

2) Split the data D of a single astrobot into training set D_{train} and test set D_{test} according to a specified ration $d = |D_{test}|/(|D_{test}| + |D_{train}|)$.

3) Build the SVM model using the training data D_{train}.

4) Evaluate the model performances using the test data D_{test}.

5) Repeat the steps 2-4 for each astrobot in the focal plane.

6) Perform the k-folds cross validation by repeating the steps 2-5 a number k of iterations and then averaging the result over k.

Chapter 3

Parity-variable prediction algorithms

The parity of an astrobot indicates the direction of rotation of the second arm of the manipulator. Therefore it is a feature that does not assume continous numerical values as the coordinates of the targets, but rather it is denoted using just two variables, one for each direction of rotation of the second arm.
The parity denotes an important mechanical characteristic of each manipulator. As a matter of fact the direction of rotation of the second arm of the manipulator has a substantial importance in the convergence of a particular astrobot's end effector to its target position. For example if we consider a generic swarm of astrobots in a focal plane with a fixed configuration of the targets assigned to each astrobot, the convergence results can be very different if we change the parity of each astrobot of the focal plane. This because the direction of rotation of the second arm plays a crucial role in the coordination process of the manipulator. A change of an astrobot's parity may led to avoid a local minimum of the decentralized navigation function.
Hence the inclusion of the parity information in our prediction problem is of particular interest. However the inclusion of a categorical information like the parity among the continous numerical informations given by the coordinates of the targets of each astrobot turns out to be a challenging problem. In fact the introduction of the parity feature inside our data samples has to be done in a smart way. On one hand we want that the information given by the parity is exploited by the predictor in the same way as the information given by the positions of the targets. On the other hand the performances of the predictor should not be driven for the most part by the information of the parity, since the information about the coordinates of the targets covers an important role in the classification as well.
Both the KNN and the SVM predictor use a distance metric for the classification of the samples. However for the parity-variable classification task it has been exploited just the SVM algorithm. Although similarity criteria exist which include also the presence of categorical variable inside the data samples [16], it has been decided to not use them in the KNN predictor. The reason of this choice lies in the fact that the distance function defined for the KNN problem would have a more ambigous meaning. Indeed it should enclose both the normal distance between the targets of the train and test configurations and the distance defined between the differents parity of the astrobots. On the other hand, the SVM predictor is more efficient in catching the nonlinearities of the data distribution, and for this reason it has been decided to include the categorical variable given by the parity inside the data samples of the SVM predictor.

3.1 SVM algorithm

As already stated in section 2.2, the SVM algorithm build a model of the data distribution by solving an optimization problem. The data samples are vectors of many features and the algorithm find a non-linear boundary between the data of the two classes. In this case we are embedding the information of the parity inside the vectors of our dataset. The definition of an astrobot $\boldsymbol{\pi}$ becomes:

Definition 11. An astrobot $\boldsymbol{\pi}$ is denoted as a column vector whose length equals the number of the parameters describing it and its neighbours:

$$\boldsymbol{\pi} = \begin{bmatrix} x \\ y \\ P \\ x_{n\ 1} \\ y_{n\ 1} \\ P_{n\ 1} \\ \vdots \\ x_{n\ r} \\ y_{n\ r} \\ P_{n\ r} \end{bmatrix} \tag{3.1}$$

Where x, y denote the x and y position respectively of the target of the central astrobot while P is a numerical value which represent the parity of the central astrobot, r indicates the number of neighbours of the astrobot. The parameters $x_{n\ i}$ and $y_{n\ i}$ with $i \in \{1, \ldots, r\}$ denote respectively the x and y position of the target of the i-th neighbour of the astrobot. Finally, $P_{n\ i}$ indicates the parity of the i-th neighbour of the astrobot.

3.1.1 Data Preprocessing

Even more than in the case of parity-based prediction, the preprocessing phase of the data plays a crucial role in order to get a good classification of the data samples. In this case the transformations applied are not the same for every feature. We have to consider that the parity is represented by a numerical element which can assume just two values, one for each direction of rotation of the second arm. On the other hand, the coordinates of the targets can take much more than two numerical values, since the position of the targets can vary widely inside the patrol zone of each astrobot. Therefore it is necessary to manage in a intellgent way the two different types of features of each sample.
The first transformation is applied to the features which represent the coordinates of the targets of the astrobots. In particular, as in the case of parity-based prediction, we scaled the range of variation of the targets to the range [-1,1] using a min-max normalization, following equation 2.18.
At this point, in order to preprocess the parity features in a reasonable way, it is necessary to make some

considerations about the standard deviation of the features corresponding to the targets coordinates. The position of the targets inside the patrol zone of each astrobot has been generated using a standard uniform distribution. However, the patrol zone can be represented geometrically by an annulus, since an inner bound is present due to the fact that the second arm is longer than the first arm. Then the targets are uniformly distributed in a interval which is not continuous. For this reason, when the coordinates are scaled inside the range [-1,1], it is not possible to apply the formula for the standard deviation of a uniform distribution. Indeed if we use that formula, the standard deviation would be 0.577. However, if we compute the standard deviation without using the formula but just using the data we have, we obtain a standard deviation that is ~ 0.5. If we use the values -1 and 1 to denote the parity, then the corresponding standard deviation of the parity features would be ~ 1. This leads to have an imbalance of the explained variances of the data. The explained variance is the ratio between the variance of a specific feature and the sum of the variances of all features of the data. Then it is important to distinguish between the variance (and standard deviation) of a given feature, which is an absolute measure since it does not depend on the variances of the other features, and the explained variance, which is a relative measure, since it depends on the distribution of the values of all the features of the data. The sum of the explained variances of all the features is equal to 100 %.

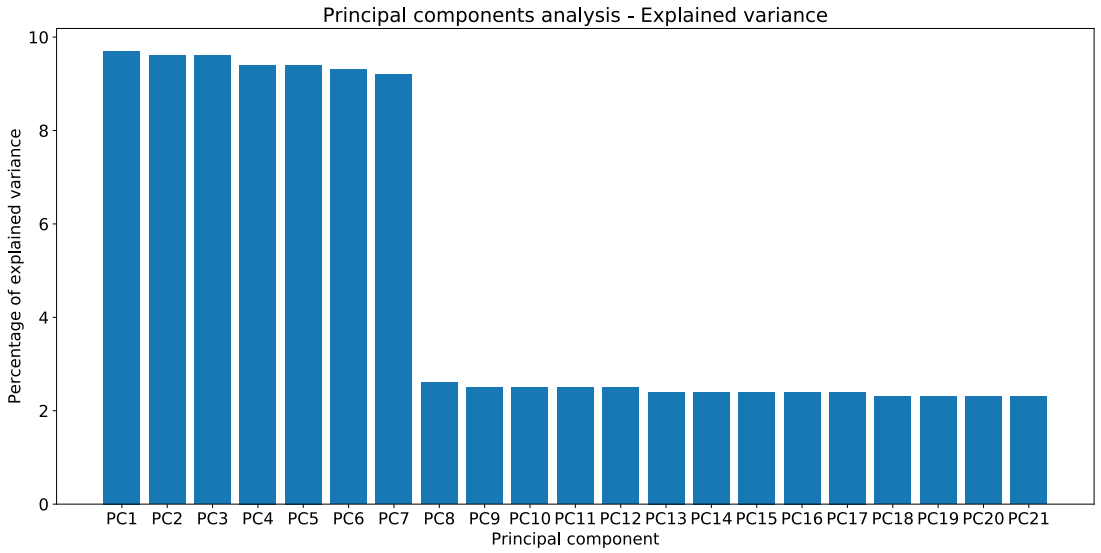


Figure 3.1: Explained variance for each component of the data of an astrobot with six neighbours. It is possible to notice that the first seven principal components have a much higher explained variance with respect to the others. These components indeed represent the parity features of the data. This means that the classification process is mainly driven by the information given by the parity. Note that in this case the neighborhood has a total of seven astrobots. Since each astrobot is denoted by three parameters, the total number of features of a data sample is 21.

If the values -1 and 1 are used for the parity, the SVM predictor would classify the new samples relying more on the information given by the parity. Indeed on average the distance between two vectors in

the feature space will be defined more by the difference between the parity features of the two vectors. Therefore the information given by the parity would cover a greater role in the classification with respect to the information given by the position of the targets. As a result the performances of the predictor would be undermined.
In order to visualize the explained variances of all the features of the data it has been performed a principal component analysis. In this case we are not interested in a dimensionality reduction of the data but rather we just want to observe how the explained variances are distributed. The term "component" is then used as synonym for feature. In fig. 3.1 are reported the explained variances of each component of the data samples for an astrobot with six neighbours.
In order to get a classifier which is driven in the same manner by the information of the parity and the information of the target's positions, it is necessary to scale the numerical values which represent the parity. The scaling of the parity feature must be done so that the resulting explained variance is more or less at the same level of the explained variance of the features which represent the target position. Then a reasonable choice is to use -0.5 and 0.5 as numerical values which indicate the two different directions of motion of the second arm. In this way the standard deviation of the parity features is $\sim$0.5, which is the same value of the standard deviation of the features corresponding to the target's coordinates. In fig. 3.2 are reported the explained variance of each component of the data for an astrobot with six neighbours in the case of a scaled parity.

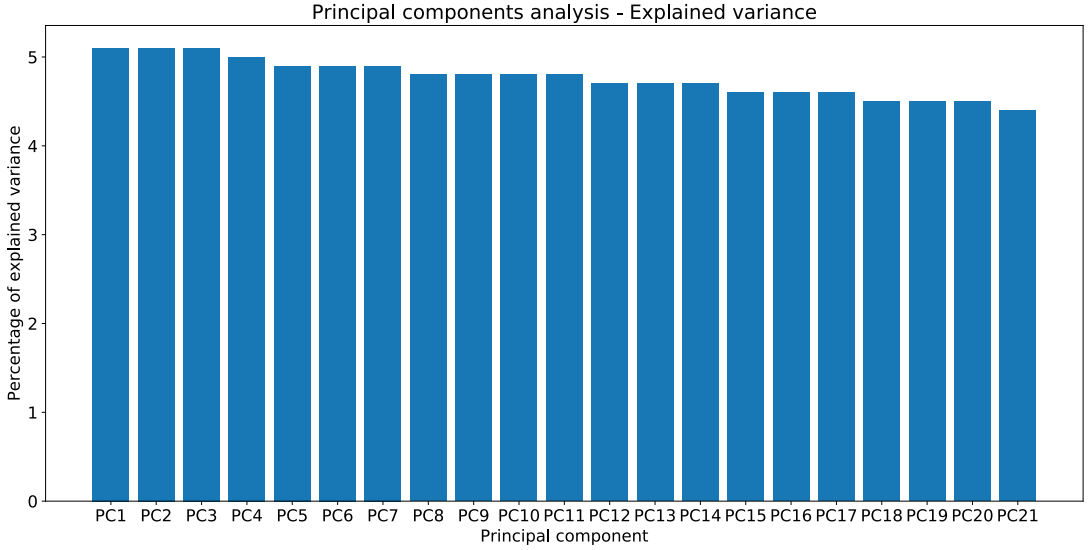


Figure 3.2: Explained variance for each component of the data of an astrobot with six neighbours. In this case the parity features are scaled to the values -0.5 and 0.5 and it is possible to see that the explained variance of each features is more or less at the same level.

Using these values for the parity, it is possible to get good performances of the predictor. If instead we chose to use lower numerical values for the parity (e.g. -0.3 and 0.3), we would have the opposite

problem, that is the model would be driven mainly by the information given by the targets, with respect to the information given by the parity.

3.1.2 SVM predictor

As in the case of parity-based prediction, the algorithm solves the optimization problem defined by the equation 2.19. The difference is that now the feature space has a greater dimension with respect to the parity-based case, since we are embedding the information given by the parity inside the vectors. Even in this case we used the Gaussian kernel in order to maps the distance between the vectors in the higher dimensional space, since we still have strong nonlinearities in the data distribution. The choice of the hyperparameters is discussed in sec. (result section for the SVM parity variable).

Chapter 4

Prediction results

In order to better understand the results of the parity-based algorithms, it is necessary to first describe the hyperparameters of the two algorithms and the performance indices used to evaluate the output of the algorithms.

4.1 Hyperparameters

KNN-based predictor

The hyperparameters of the KNN-based predictor are:

- The number of closest train configuration K. If we increase this number we increase the number of closest train configuration to take into account when computing the output probability vector $\mathbf{p}$.

- The corrector coefficients α and β. These two parameters are introduced to manually tune the weights w_i of the vector $\mathbf{w}$, in order to get better accuracy of the positives or of the negatives. In particular, α tunes the weights w_i for the astrobots in a full neighborhood configuration, while β tunes the weights w_i of the astrobots which are not in a full neighborhood configuration.
  A choice of $\alpha = \beta = 1$ means that the weights w_i of the vector $\mathbf{w}$ are not modified.

- The distance metric d. With the letter d we indicate a generic distance metric. The metric chosen in our case is the metric d_1 which has already been discussed in sec. 2.1.3.

SVM-based predictor

The hyperparameters of the SVM-based predictor are:

- The type of kernel function κ. In our case we will use a gaussian kernel as indicated in the equation 2.20 but there are many other types of kernel functions (polynomial, sigmoid, ecc.). We chose to

adopt the Gaussian kernel because it is the one that yields better results in the prediction. Indeed the polynomial or the sigmoid kernels functions are suitable when the data are not linearly separable but there is a sort of scheme in the data distribution which can be catch by a well defined function. In our case there is not clear pattern in the distribution of data of the two classes. The use of a Gaussian kernel function allows to better cluster the data when there are strong nonlinearities in the data distribution.

- The kernel size σ. This parameter determines the width of the gaussian kernel function. The kernel size determines the precision of the fit given by the nonlinear hyperplane. In particular, if σ is too small this means that the gaussian kernel is very narrow and the boundary may be not able to separate one of the two classes. The same can happen if the kernel size is very high, that is the hyperplane is not able anymore to fit the two classes in a correct manner because the the boundary loses its characteristics of non-linearity.
  In order to have an idea of the order of magnitude of the kernel size, it is necessary to measure the average euclidean distance between the vectors of the dataset, since in the equation 2.20 the square of the euclidean distance appears in the exponent's numerator.

- Missclassification penalty C. This parameter regulates the importance of avoiding misslcassification of the training samples. In particular, a large C implies that we are highly penalizing missclassification of the training samples. The consequence consists in having a smaller margin for the boundary (see Fig. 4.1). On the contrary, if we have a small C, we are allowing more training samples which are not correctly classified in the traing phase and the margin of the hyperplane will be larger [14].

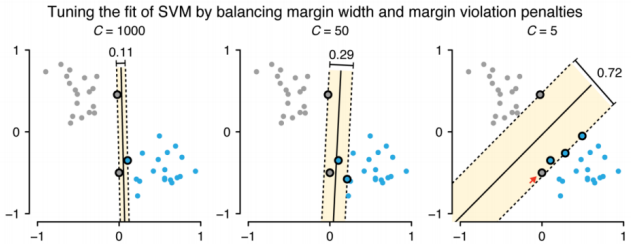


Figure 4.1: Effect of the hyperparameter C for linearly separable data. The same effect on the width of the margin holds for data which are not linearly separable (Reprinted with permission from [17]).

There is not a rule of thumb for the choice of C. It depends on the dataset we are using for our machine learning problem. It is important to underline that using a large missclassification penalty for trainig data does not ensure that the missclassification errors for testing data will be low. For this reason it is not possible to increase at will the value of C.

- Class weight w_1. This parameter is stricly related to the choice of the missclassification penalty C. Indeed, in the case of imbalanced data classes, the missclassification penalty C_j for the j-th class is computed as follows:

$$C_j = C \cdot w_j \tag{4.1}$$

Where C is the missclassification penalty and w_j is the weight of the j-th class. In our case we have chosen to modify the class weight of the positive class (the class of 1s), which is the majority class. Since the 1s are more frequent with respect to the 0s, the class weight w_1 has a value which is less than 1, indicating that we are decreasing the weight given to the majority class when solving the optimization problem. The weight of the minority class is fixed to a value of 1.
It is important to underline that is possible to do also the opposite, that is to tune the weight w_0 of the minority class and keeping fixed the value of w_1 to 1. In that case w_0 should have a weight bigger than 1, since the class of negatives is less frequent with respect to the class of positives. In that case C has to be tuned as well in order to achieve the same results as in the case where w_0 is fixed and w_1 is variable.

A good choice for the hyperparameter C has been found by performing a manual tuning, while good values for the hyperparameters σ and w_1 have been found by performing a grid search, keeping fixed the value of the missclassification penalty C previously obtained.
Since for the SVM algorithm we train a model for a single astrobot, the hyperparameters will change according to the number of neighbors of each astrobot.

4.2 Simulations setup

The simulations have been performed on a Dell Inspiron 15 7000 with a processor Intel Core i7-7700HQ and a CPU with 2.80 GHz as clockspeed. The RAM has 16 GB and the operating system is Windows 10 Home 64 bit.

KNN-based predictor

The KNN-based predictor has been developed as a script on Matlab R2019b. The script is structured in four main sections:

- Loading the data
- Data analysis
- KNN-based predictor
- Analysis of the results

The first part consists in loading the data of our dataset. The data are collected in a ".csv" format, but once loaded in the script they are converted in ".mat" format.
The second section performs an analysis of the data. In particular are computed the overall percentage of convergence of all the astrobots in the focal plane, then the percentage of convergence of the single astrobots in the focal plane, and finally the number of neighbours of each astrobot in the focal plane.
The third part represents the core of the algorithm. Before executing this part, the user can set the dimensions of the test dataset D_{test} and the training dataset D_{train}, the number of iterations k of the cross validation, the coefficients α and β and the number of closest train configurations K to take into account when computing the output of the prediction.
The predictor consists in two nested loop. The outer loop is repeated a number k of times and it represents the cross validation process. The inner loop computes the prediction of convergence given a test configuration $\mathbf{T}$. This loop is repeated a number N of times, where N is the number of test configurations in the test dataset D_{test}. Inside this loop is performed the local analysis, it is computed the binary output of the prediction and then the binary output is compared with the ground truth vector of the test configuration, in order to obtain the number of true positives, true negatives, false positives and false negatives.
The final section of the algorithm allows the user to visualize the performance of the predictor by using some indices and diagrams which will be explained in section 4.3.

SVM-based predictor

The SVM-based predictor has been developed as a Python script. The SVM model has been built using the Python library Scikit-Learn [18]. This library allows to build an SVM model given the hyperparameters of the prediction.
The script can be divided into two main parts:

- SVM predictor
- Analysis of the results

Even in this case we have two nested loop. The outer loop represents the cross validation process (as in the case of KNN). This loop is repeated a number k of times. The inner loop is needed to build and evaluate the SVM model. In particular this loop is repeated a number r of times, where r indicates the number of astrobots in the focal plane. At each iteration of this loop a data file in ".csv" format corresponing to the data of a single neighborhood of the focal plane is loaded and transformed into a multidimensional array. Then the data are divided into training data and testing data. At this point the user can set the hyperparameters of the classifier, depending on how many neighobours are present in the particular neighborhood under analysis. In fact, since in this case the model is generated for a single astrobot's neighborhood, it is important to specify the correct hyperparameters according to the number of neighbors present in the neighborhood.
Then the model is evaluated using the testing data. The output of the classifier is compared with the ground truth of the test samples in order to obtain the number of true positives, true negatives, false positives and false negatives.
The second part of the script allows the user to visualize the performances of the classifier using the indices and the metrics described in section 4.3.

4.3 Perfomance metrics

Before describing the metrics which are used to evaluate the results of the prediction, it is necessary to give some definitions:

Definition 12. A true positive (TP) is an astrobot that is predicted to converge (predictor predicts 1) and it actually converges to its target position (its corresponding ground truth element is 1).

Definition 13. A false positive (FP) is an astrobot that is predicted to converge (predictor predicts 1) but it actually does not converge to its target position (its corresponding ground truth element is 0).

Definition 14. A true negative (TN) is an astrobot that is not predicted to converge (predictor predicts 0) and it actually does not converge to its target position (its corresponding ground truth element is 0)

Definition 15. A false negative (FN) is an astrobot that is not predicted to converge (predictor predicts 0) but it actually converges to its target position (its corresponding ground truth element is 1).

The true positives, the true negatives, the false positives and the false negatives are the result of the comparison between the output of the prediction and the ground truth of the test data we are using to evaluate the algorithm.

Definition 16. The true positive rate (TPR) is the ratio between the true positives of the prediction and the total number of positives:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{4.2}$$

Where P indicates the total number of positives, which is the sum of the true positives and the false negatives.

Definition 17. The true negative rate (TNR) is the ratio between the true negatives of the prediction and the total number of negatives:

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{4.3}$$

Where N indicates the total number of negatives, which is the sum of the true negatives and the false positives.

From an engineering point of view we are more interested in the correct prediction of the positives (the astrobots which converge to their target position), since the information on the prediction of these astrobots would be crucial when choosing whether or not to start the coordination process for a particular configuration of the targets in the focal plane.
On the other hand we cannot have only the information on the correct prediction of the 1s as performance index for our predictor. This because the number of positives is much greater with respect to the number of negatives. For example, if our predictor predicts always 1, we would have a perfect TPR but our "predictor" is not actually a predictor, since it is always predicting 1, without caring on the presence of the 0s.
The problem of the unbalanced data classes is also responsible for the use of the definition of the balanced accuracy instead of the classic definition of the accuracy.
The classic definition of accuracy is the following:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.4}$$

Since the data classes are not balanced, if we have a predictor which predicts always 1 the accuracy would be still high.
Instead of using the classic definition of the accuracy it is then necessary to use the balanced accuracy, which is defined as follows:

$$BA = \frac{TPR + TNR}{2} \tag{4.5}$$

In this definition we are giving the same weight to the correct prediction of the 1s and to the correct prediction of the 0s. With this definition it is possible to have a more accurate information on the performence of the algorithm.

Confusion matrix

The confusion matrix is a table which allows to visualize the performance of the classifier. Since we are performing a binary classification, our confusion matrix has just two rows and two columns. The matrix allows to rapidly understand how many samples have been correctly classified and how many samples have been assigned to the wrong class (see Fig. 4.2)

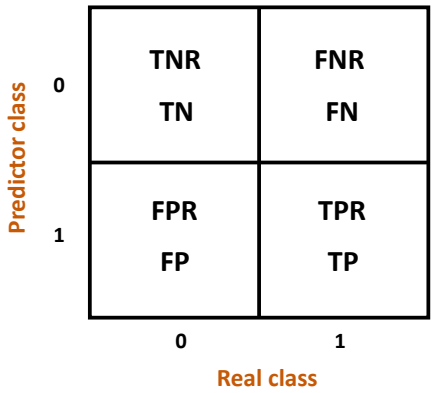


Figure 4.2: Confusion matrix. The notations TN, FN, FP and TP indicate the number of true negatives, false negatives, false positives and true positives respectively. While TNR, FNR, FPR and TPR indicate the true negative rate, the false negative rate, the false positive rate and the true positive rate respectively, all expressed as percentages.

If a predictor shows good performances, the corresponding confusion matrix has high values on the main diagonal, indicating that the majority of samples have been correctly classified.

ROC curve

The Receiver Operating Characteristic (ROC) curve illustrates the performance of a classifier when varying one of the parameters of the predictor. On the y-axis is reported the true positive rate (TPR), while on the x-axis is reported the false positive rate (FPR) which is defined as 1 - TNR.
A perfect classifier has a TPR which is maximum and a FPR which is minimum (because the TNR is maximum), then the point is located on the top left of the graph (see Fig. 4.3).

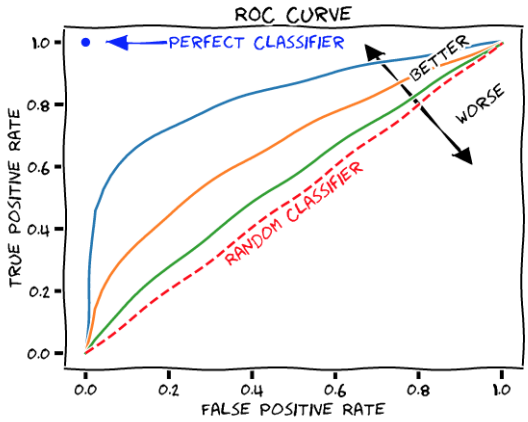


Figure 4.3: Example of differents ROC curve (Reprinted with permission from [19]).

The dotted line which connects the bottom left angle of the graph with the top right angle is called "random guess line". The more our curve is close to this line, the more our predictor is just "guessing" the output of a test sample. While the more the curve is above the random guess line, the more the predictor is performing well [20].

Precision

The precision is defined as:

$$P = \frac{TP}{TP + FP} \tag{4.6}$$

At the denominator of the formula we have the total number of predicted positives. The precision indeed is an index of how accurate the predictor is in predicting the positives.
Precision is an important measure to look at when false positives have a significant role in our problem. In our case a false positive is an astrobot which is predicted to converge but it actually does not converge to its target position. Our goal is then try to minimize the false positives and increase the precision.

F1 score

The F1 score is a metric which is used when we want to look at the trade off between precision and true positive rate. The true positive rate (TPR) in also indicated in literature with the term "Recall" (R).
The formula for the F1 score is:

$$F1 = \frac{2 \cdot R \cdot P}{R + P} \tag{4.7}$$

In our problem it is very difficult to increase the recall and the precision at the same time. For example, if we increase the TPR we are increasing the number of true positives predicted, but in all probability we will also increase the number of false positives, decreasing the precision.
For this reason the F1 score can be a useful indicator for our problem. The bigger is the F1 score, the better is our trade off between precision and recall [21].

4.4 Parity-based prediction results

In this section are reported the results of the parity-based prediction algorithms, which are the KNN-based algorithm and the SVM-based algorithm.

4.4.1 KNN results

Before illustrating the results of the algorithm, it is necessary to describe how the results were obtained. Suppose we have a testing dataset of N configurations $\mathbf{T_i}$ with $i \in \{1,2,\dots,N\}$ and a number k of iterations of the algorithm.
Let's indicate respectively with TP_i, TN_i, FP_i and FN_i the number of true positives, true negatives, false positives and false negatives of a single test configuration $\mathbf{T_i}$, which are obtained comparing the ground truth vector $\mathbf{t_i}$ of the test configuration $\mathbf{T_i}$ with the output of the prediction $\mathbf{y_i}$. Let's also indicate with r a generic iteration of the algorithm, it follows that $r \in \{1,2,\dots,k\}$.
At the end of one iteration of the algorithm we get a total number of true positives TP_r, true negatives TN_r, false positives FP_r and false negatives FN_r which are computed as follow:

$$TP_r = \sum_{i=1}^{N} TP_i \quad TN_r = \sum_{i=1}^{N} TN_i \quad FP_r = \sum_{i=1}^{N} FP_i \quad FN_r = \sum_{i=1}^{N} FN_i \tag{4.8}$$

Then at every iteration r of the algorithm we compute the total number of true positives, true negatives, false positives and false negatives for all our testing dataset, according to the equations in 4.8. The final results are the average over the number of iterations of the TP_r, TN_r, FP_r and FN_r:

$$TP = \frac{\sum_{r=1}^{k} TP_r}{k} \quad TN = \frac{\sum_{r=1}^{k} TN_r}{k} \quad FP = \frac{\sum_{r=1}^{k} FP_r}{k} \quad FN = \frac{\sum_{r=1}^{k} FN_r}{k} \tag{4.9}$$

Where TP, TN, FP and FN indicate respectively the final number of true positives, true negatives, false positives and false negatives of the prediction experiment. All the metrics described in sec. 4.3 are computed using these values.
In the following analysis it is important to keep in mind that we indicate with K the number of closest train configurations to take into account when computing the output of the prediction, while we indicate with k the number of iterations of the algorithm.

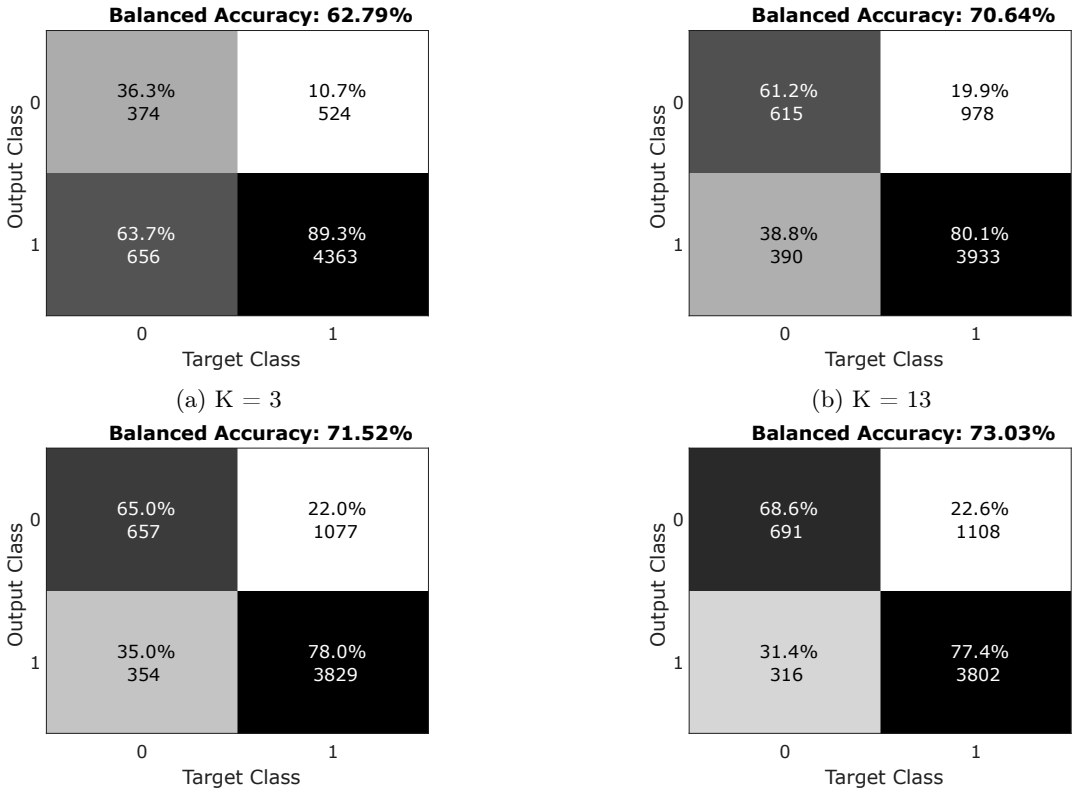
Focal plane with 116 astrobots

The complete dataset is composed of 10100 configurations. Since the KNN is a memory-based learning algorithm, the more configurations we have in the training set, the more accurate will be the prediction. For this reason our testing set is composed of just 51 configurations, while the training set has 10049 configurations.

Number of neighbours of the astrobot	Number of astrobots
6	57
5	24
4	29
3	6
2	0

Table 4.1: Neighbours distribution for a focal plane with 116 astrobots

In this focal plane, the number of neighbors for each astrobot is distributed according to the table 4.1.



(a) K = 3

(b) K = 13

(c) K = 25

(d) K = 39

Figure 4.4: Confusion matrices for the KNN predictor tested on a focal plane with 116 astrobots. The two corrector coefficients α and β are both equal to 1 in each case. It is possible to observe that the TNR increases with K, while the TPR deacreases with K. This is reasonable since the more train configurations we take into account for the computation of the output, the more is likely to consider train configurations which have astrobot that don't converge.

The number of iterations k of the algorithm has been set at 15. In any case, also tests were carried out with a higher number of iterations to certify the reliability of the results. But since in this case

the cross validation requires a considerable amount of time, it was decided to adopt a not too high k, especially when tests were performed to observe the effect of hyperparameters in the prediction.
In Fig. 4.14 are reported the confusion matrices for different values of K (number of closest train configurations), keeping fixed at 1 the two corrector coefficients α and β.

The choice of the number of closest train configurations K depends on how big is our training dataset. The more the training dataset is large, the more we can choose a larger value of K. In our case, with a training dataset with 10049 configurations a good choice of K is comprised between 10 and 50. If we increase K too much the information on the targets location of the closest train configuration is not reliable anymore.
The results shown in Fig. 4.14 are for the entire focal plane. It is interesting to compute the performance indices for the single astrobots. In particular we want focus our attention on the number of neighbors of each astrobot, and computing the accuracy results on the basis of this information.
According to the table 4.1 there are 57 astrobots in the focal plane which are surrounded by other six astrobots. This means that the results of the prediction on the entire focal plane depend largely on the performances of the astrobots with six neighbours.

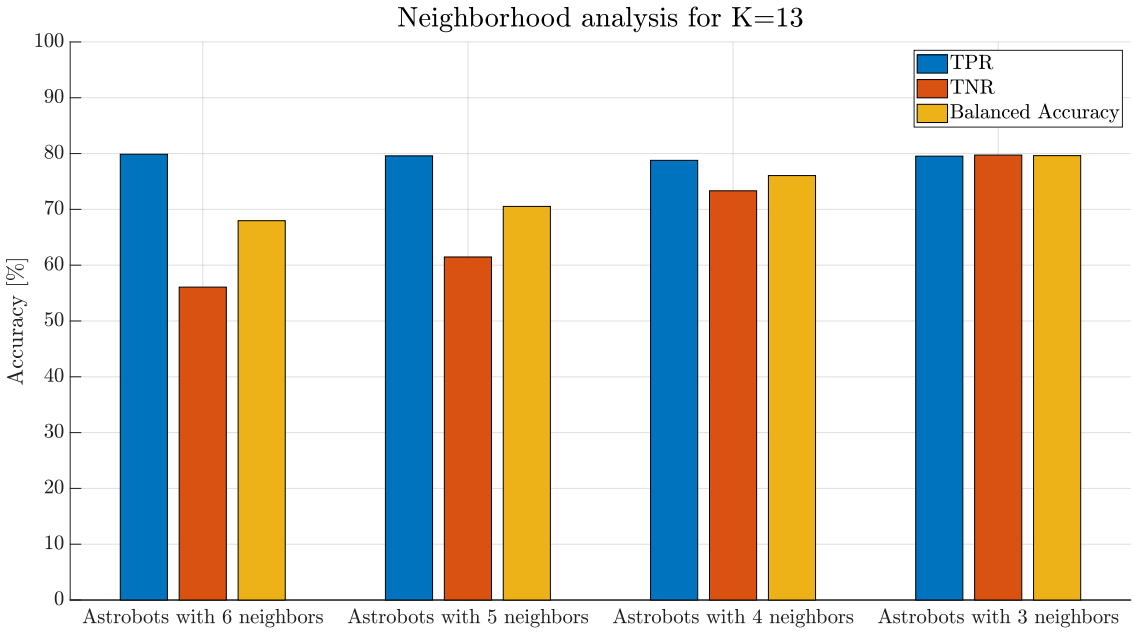


Figure 4.5: Average TPR, TNR and *Balanced Accuracy* for the differents astrobots in the focal plane, depending on the number of neighbours of each astrobot. It is possible to notice that the less neighbors astrobots are present, the more the performances increase. In particular the most critical condition for prediction is for an astrobot with six neighbors.

It is possible to compute the TPR, TNR and the *Balanced accuracy* of the single astrobot in the focal plane. In order to analyze the performances of the astrobot depending on the number of neighbors, we take the average of the performances of the astrobots with a specific number of neighbors.
For example, let's indicate with N the number of astrobots with six neighbours in the focal plane and with $TPR_{n6\ i}$ the true positive rate of an astrobot with six neighbours, $i \in \{1,2,\dots,N\}$. The average TPR of the astrobots with six neighbours is then computed as:

$$TPR = \frac{\sum_{i=1}^{N} TPR_{n6\ i}}{N} \tag{4.10}$$

In the same way we can compute the TNR and the *Balanced accuracy* of the astrobots with six neighbours, and then repeat the procedure for the astrobots with five, four and three neighbours.
In Fig. 4.5 and Fig. 4.6 are reported the results of the neighborhood analysis for K=13 and K=39 respectively, keeping always the two correctors coefficients α and β equal to 1.

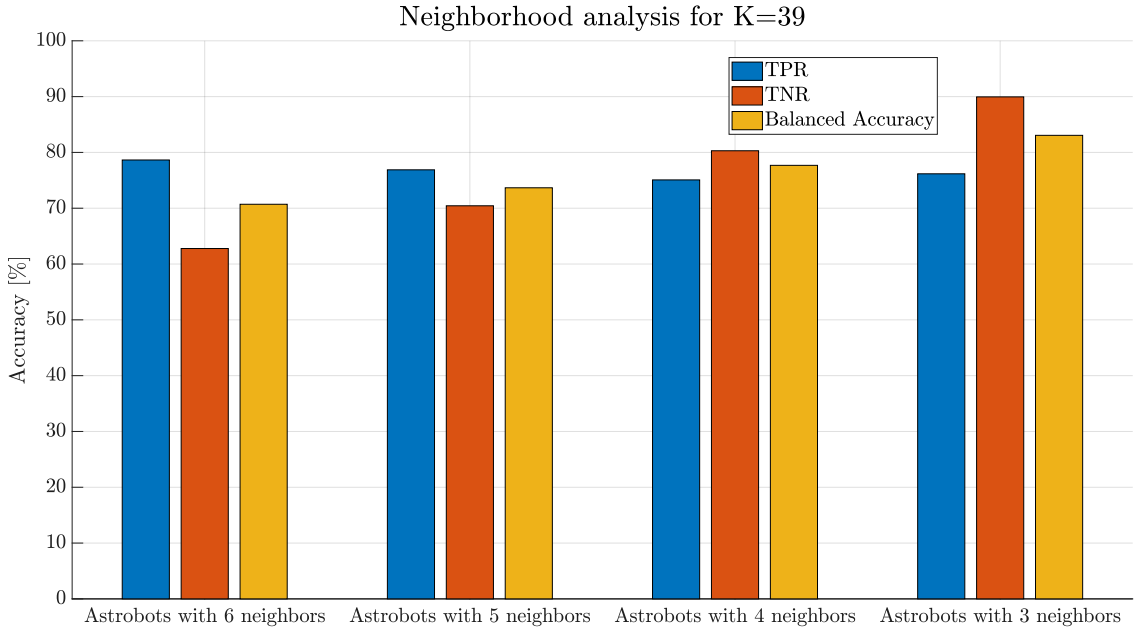
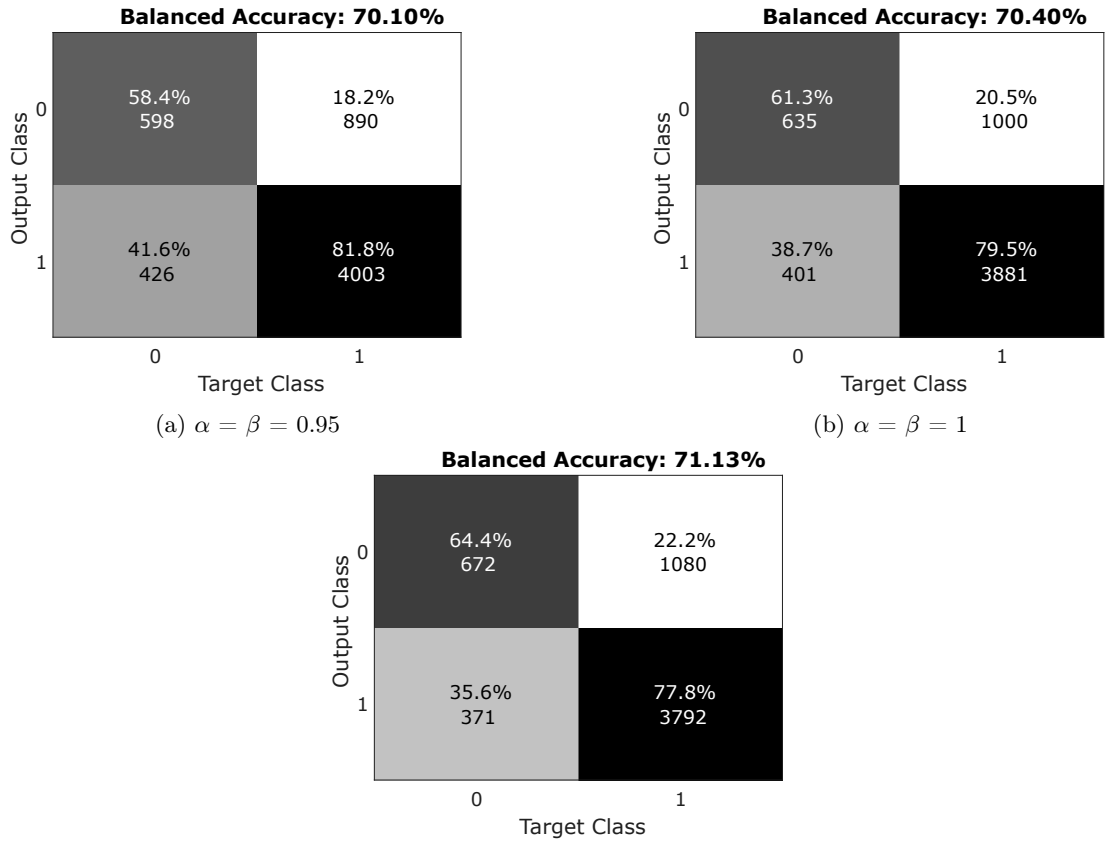


Figure 4.6: Average TPR, TNR and *Balanced Accuracy* for the differents astrobots in the focal plane, depending on the number of neighbours of each astrobot. In this case we can observe an improvement in the balanced accuracy of the astrobots with six neighbours. On the other hands the astrobots with less than six neighbours show a decrease of the TPR and an increase of the TNR.

The Figures 4.5 and 4.6 provide very indicative information regarding the performances of the individual astrobots, depending on the number of neighbors. The worst condition for prediction is for an astrobot with six neighbours. For this reason the two coefficients α and β have been introduced. The possibility to change the weights of the minority class in a separate manner for the astrobots with six neighbours and the ones with less than six neighbours will be useful in order to improve the

performance of the predictor.

The coefficient α multiplies the weights w_i of the vector $\mathbf{w}$ computed using the equation 2.6 only for the astrobots which have six neighbours. While the coefficient β multiplies the weights w_i of the vector $\mathbf{w}$ only for the astrobots which have less than six neighbours. This means that if α and β are bigger than 1 we are increasing the weight given to the 0s. If they are less than 1, we are decreasing the weight given to the 0s.
In Fig. 4.7 are reported the confusion matrices of three different predictions when varying the two corrector coefficients. In this case they have been kept equal to each other just to show the overall effect of increasing the weight of the 0s in the prediction. In the three cases the number K was equal to 13.



(a) $\alpha = \beta = 0.95$

(b) $\alpha = \beta = 1$

(c) $\alpha = \beta = 1.05$

Figure 4.7: Confusion matrices for predictions with differents coefficients α and β. In each case the number K has been kept fixed to 13. It is possible to observe an increase of the TNR and a decrease of the TPR when the two coefficients increase. This is a direct effect of increasing the weight of the minority class.

The way in which the number K and the coefficients α and β affect the prediction can be visualized in the Fig. 4.8 and 4.9. In these pictures each point on the graph represent the result of a simulation performed using the correspondent hyperparameter on the horizontal axis.

At this point, in order to find best hyperparameters for the prediction, a manual tuning has been done using as reference the graphs in the figures 4.8 and 4.9.

Since we are more interested in a correct prediction of the positives, a possible idea for the selection of K could be to increase as much as possible the TPR, making sure that the balanced accuracy does not drop below a certain treshold. For example, in this focal plane the treshold for the balanced accuracy could be set to 70%. In this way we ensure decent performances of the predictor also when predicting negatives. The best result are reported in table 4.2.

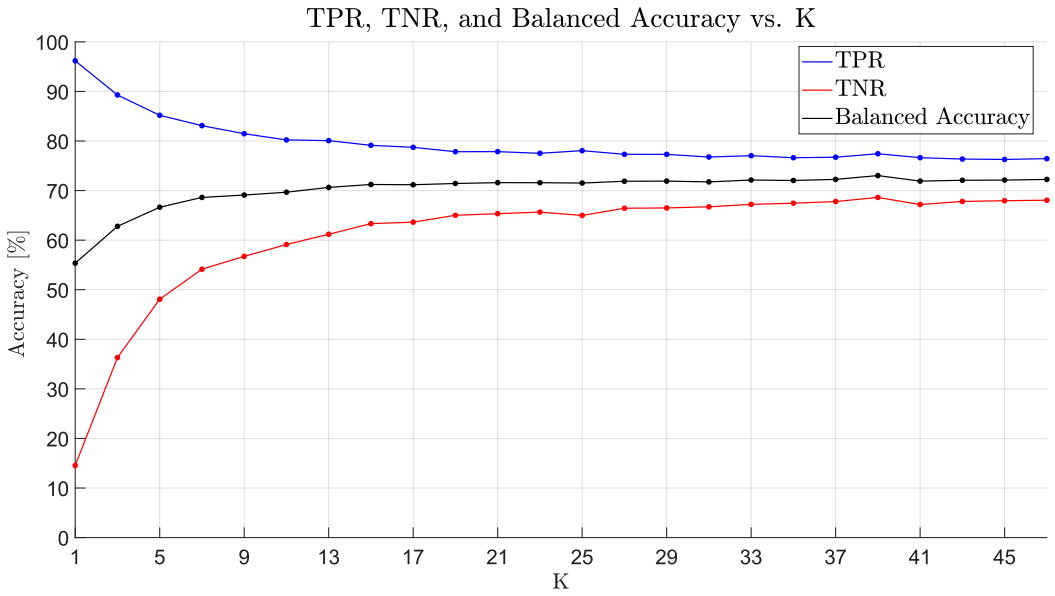


Figure 4.8: TPR, TNR and *Balanced Accuracy* plotted for differents values of K. The value of the two coefficients α and β is equal to 1. It is possible to observe that performances are stable for K greater than 19. For K = 39 there is a little peak in terms of performances. Then a reasonable value for the choice of K could be 39 for the KNN prediction in this focal plane.

From figure 4.8 it is possible to observe that we can have more than one good value of K depending on what is our focus in the prediction. For example, if we want increase the balanced accuracy as much as possible, allowing a decrease of the TPR below 80%, then K=39 is probably the best choice. But if we want to keep the TPR above 80 % and try to maximize as much as possible the balanced accuracy, then the best choice is K=13.
This is to highlight once again how the choice of the hyperparameters is stricly correlated to what is our prediction goal.
In fig. 4.9 is possible to observe the trends of TPR, TNR and balanced accuracy when varying the two coefficient α and β. The value of K has been kept fixed to 13.

In in fig. 4.10 it is possible to visualize the difference of TPR for the astrobots with a different number of neighbours when varying the coefficient β, while in fig. 4.11 is reported the difference of TPR for the astrobots with a different number of neighbours when varying the coefficient α.

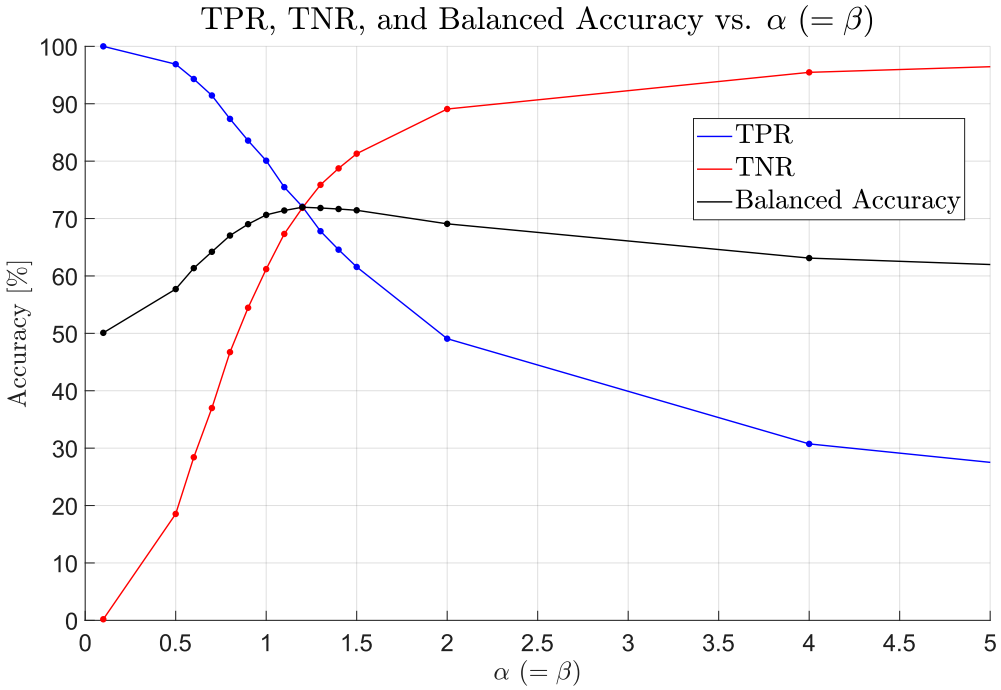


Figure 4.9: TPR, TNR and *Balanced Accuracy* plotted for differents values of the coefficients α and β. For simplicity the two coefficients have been kept equal to each other. The value of the number K is 13. It is possible to observe how the correct prediction of negatives increases with the two coefficients, this is reasonable since we are increasing the weight given to the 0s. On the contrary, the correct prediction of positives increases when the two coefficents decrease, since we are decreasing the weight of the 0s in the prediction.

K	α	β	**TPR (Recall)**	**TNR**	**Bal. Accuracy**	**Precision**	**F1 score**
39	1	1	77.2 %	68,51 %	72.85 %	92.22 %	84.04 %
39	1	0.85	80.44 %	63.23 %	71.83 %	91.4 %	85.57 %
31	1	0.9	79.3 %	64.7 %	72 %	91.51 %	84.97 %

Table 4.2: Best results obtained with the KNN-based predictor on a focal plane with 116 astrobots. It is possible to observe that if we decrease the coefficent β, we obtain an increase of the TPR. The increase is due to astrobots with less than six neighbors. It is possible to visualize this effect in fig. 4.10.

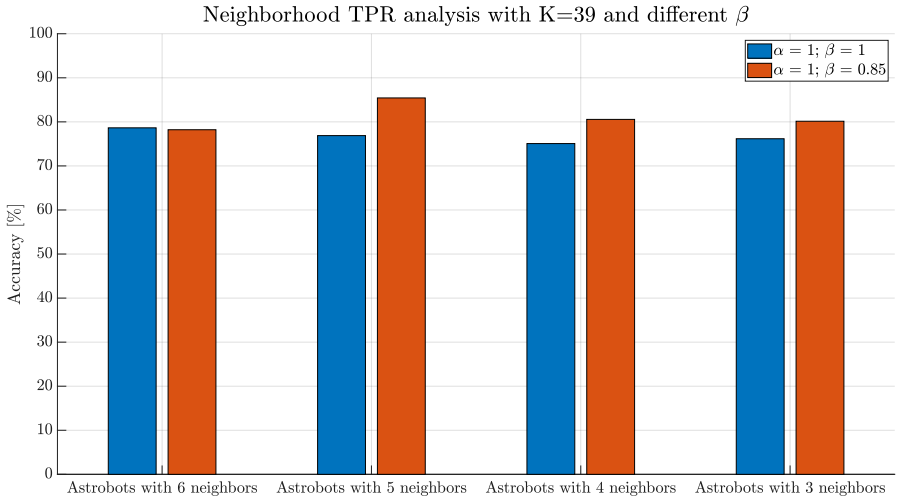


Figure 4.10: TPR for astrobots with different neighbours when varying the coefficient β and keeping fixed α and the number K. When the coefficient β is lower than 1 we are decreasing the weight given to the 0s, and then the TPR of the astrobots with less than six neighbours increase. While since α is 1 in both cases, we can observe that there is practically no difference between the TPR of the astrobots with six neighbours in the two predictions.

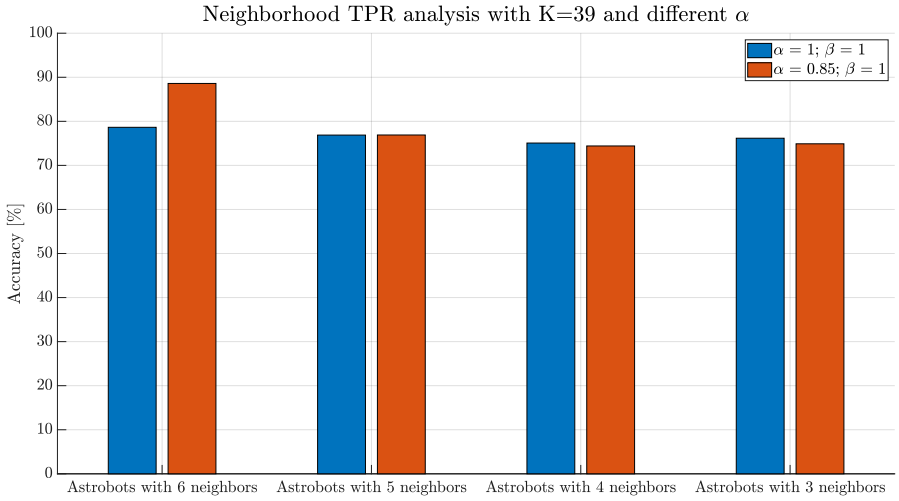


Figure 4.11: TPR for astrobots with different neighbours when varying the coefficient α and keeping fixed β and the number K. When the coefficient α is lower than 1 we are decreasing the weight given to the 0s, and then the TPR of the astrobots with six neighbours increase. While since β is 1 in both cases, we can observe that there is practically no difference between the TPR of the astrobots with less than six neighbours in the two predictions.

The fig. 4.12 shows the trend of the precision, the recall and the F1 score for differents values of K. The precision tells use how able is the predictor in correctly predicting the positives, while the recall (that is the TPR) indicates how many positives are correctly predcited over the total number of positives.

Finally we can look at the ROC curve to visualize the performances of our predictor. Every point on the ROC represents the result of a prediction experiment using a different value of one of the hyper-parameters. In fig. 4.13 is reported the ROC curve built using different values of the two coefficients α and β (set equal to each other) and with a value of K equal to 13. Again it is important to bear in mind that the choice of the coefficients α and β depends on what is our prediction goal. For this reason, taking the coefficients corresponding to a point on the curve as far as possible from the random guess line may not always be the best choice.

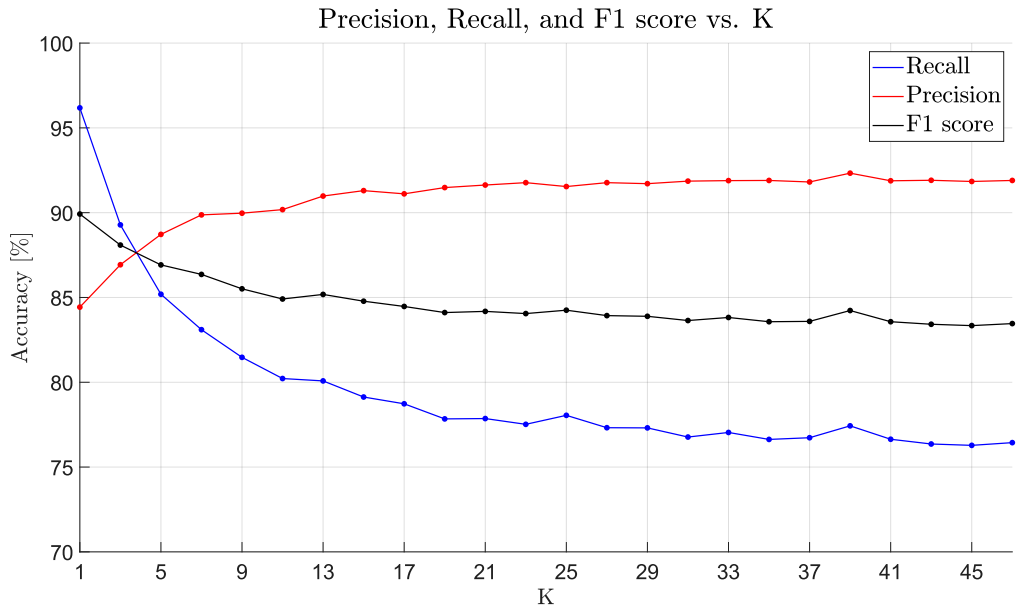


Figure 4.12: *Precision, Recall* and *F1 score* plotted for differents values of K. The best precision is reached once again for K = 39. The F1 score show the trade-off between the two indices of recall and precision. The best values for the recall are for a low value of K, but from the fig. 4.8 we know that for small values of K we have a low TNR and then a low balanced accuracy. For this reason we have to look for larger value of K, and again K=39 seems to be a good choice.

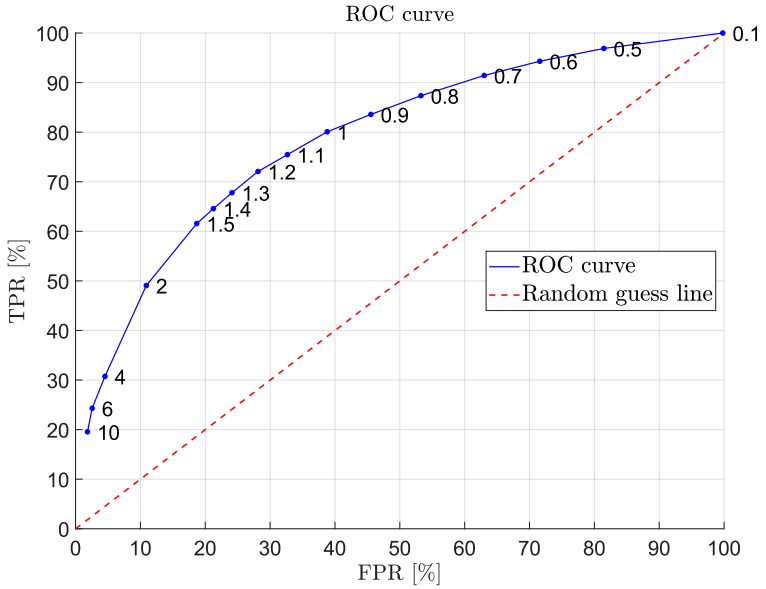


Figure 4.13: ROC curve for differents values of the coefficients α and β. The values of the coefficients for every prediction are reported near the points on the curve. The fact that our curve is above the random guess line means that the predictor is actually trying to predict the output of the samples on the basis of the informations of our training data, and not just guessing the result in a random manner (see fig 4.3).

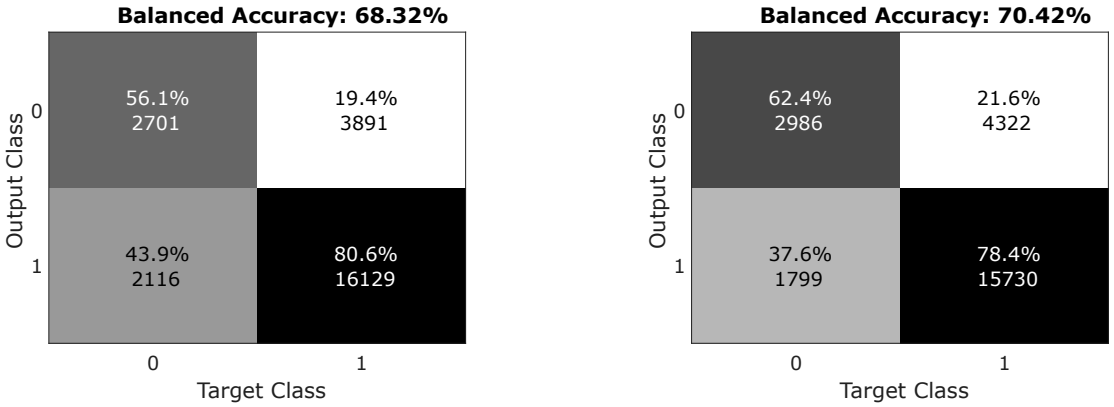
Focal plane with 487 astrobots

The complete dataset is composed of 10100 configurations. Also in this case the number of test configurations is 51, while the number of training configurations is 10049. The number of iterations k is set to 15.
In this focal plane, the number of neighbours for each astrobot is distributed according to the table 4.3

Number of neighbours of the astrobot	Number of astrobots
6	345
5	66
4	67
3	8
2	1

Table 4.3: Neighbours distribution for a focal plane with 487 astrobots

From table 4.3 it is possible to observe that most of the astrobots in the focal plane are astrobots in a full neighborhood configuration. This means that the results for the entire focal plane will depend substantially on the prediction performances for the astrobots in a full nighborhood configuration.



(a) K = 13 (b) K = 39

Figure 4.14: Confusion matrices for the KNN predictor tested on a focal plane with 487 astrobots. The two corrector coefficients α and β are both equal to 1 in each case. Again, it is possible to notice that a greater number of K leads to a better classification of the negatives. Furthermore, the balanced accuracy has lower values with respect to the case with a focal plane with 116 astrobots. This can be explained looking at the fig. 4.15 and considering that in this focal plane the 70 % of the total number of astrobots consists of astrobots in a full neighborhood configuration (worst condition for prediction).

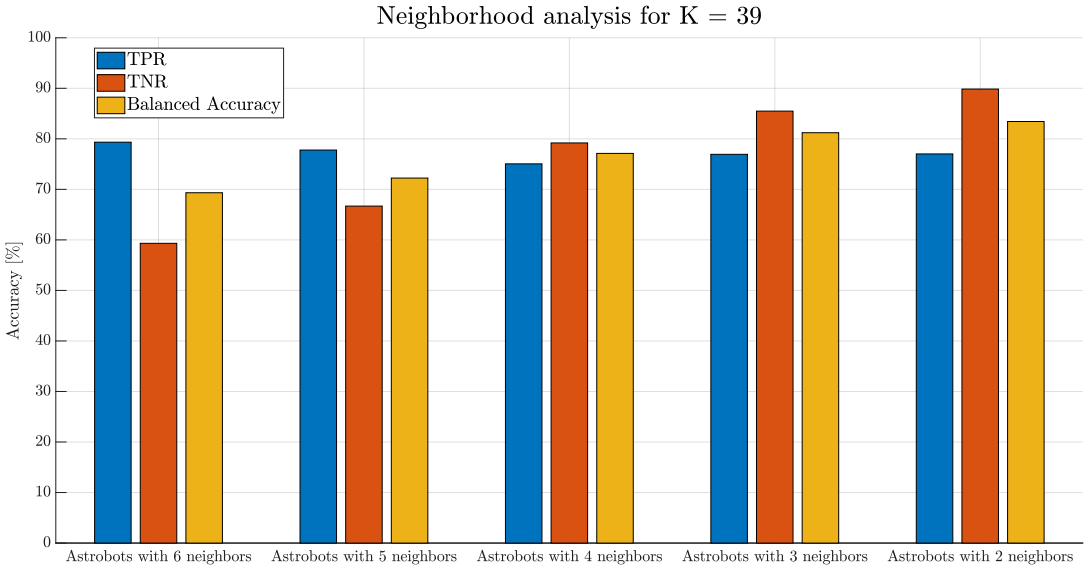


Figure 4.15: Neighborhood analysis in a focal plane with K=39 and α, β equal to 1. The worst condition for prediction is for an astrobot with six neighborhood. Since this focal plane has the majority of astrobots in a full neighborhood configuration, the prediction results on a complete focal plane will largely depend on the performances of the astrobots with six neighbours.

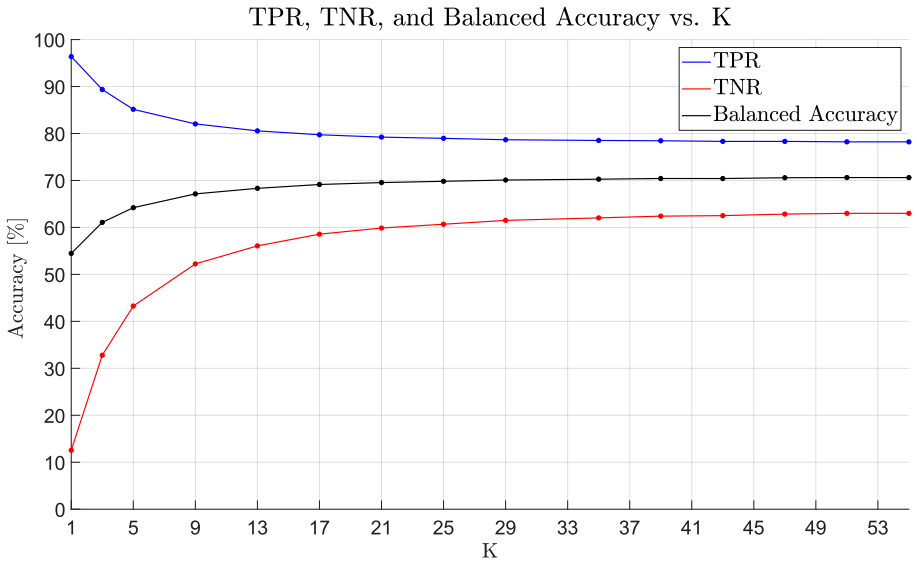


Figure 4.16: *TPR*, *TNR* and *Balanced Accuracy* for differents values of K in a focal plane with 487 astrobots. The two coefficients α and β are equal to 1. Also in this case it is possible to observe that the performances are stable for K grater than 21.

From fig. 4.16 it is possible to notice that the algorithm's performance are particularly stable when K is greater than 21. Therefore, K values in the range between 21 and 55 ensure good results if our goal is to keep the balanced accuracy above 70% and maximize the TPR. While if we want for example keep the TPR above 80% and maximize the balanced accuracy as much as possible, then a good choice is K=13.
In fig. 4.17 are reported the trends of the TPR, TNR and *Balanced Accuracy* when varying the two corrector coefficients α and β. The value of K has been kept fixed to 13, so as to allow a comparison with the case of a focal plane with 116 astrobots in fig. 4.9.
The best results for this focal plane are reported in table 4.4.

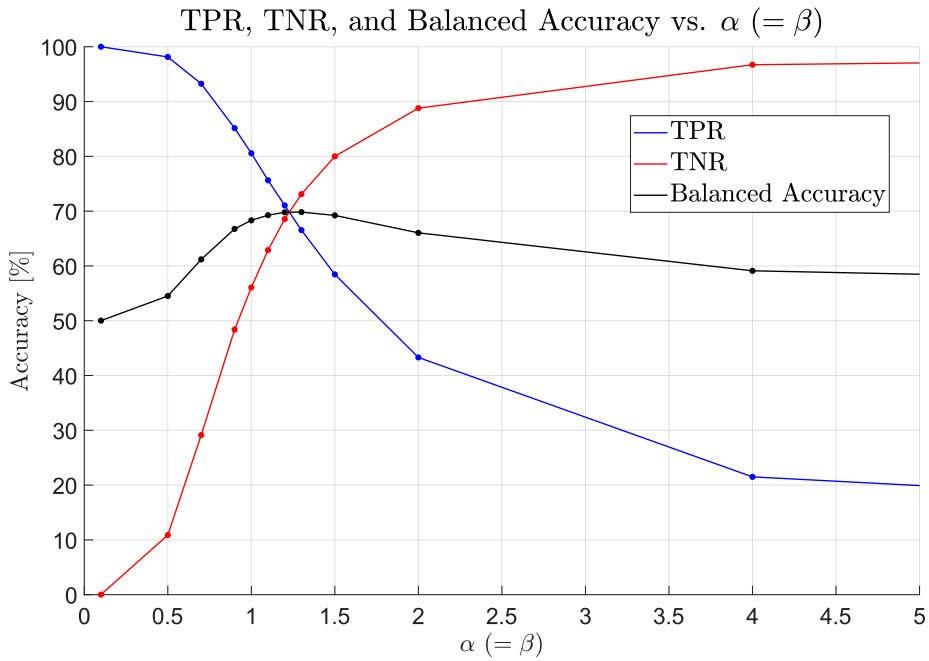


Figure 4.17: TPR, TNR and *Balanced Accuracy* plotted for differents values of the coefficients α and β. For simplicity the two coefficients have been kept equal to each other. The value of K is 13. The trends are vary similar to the case for a focal plane with 116 astrobos (see fig. 4.9)

K	α	β	**TPR (Recall)**	**TNR**	**Bal. Accuracy**	**Precision**	**F1 score**
51	1	1	78.23%	63 %	70.62 %	89.78 %	83.61 %
51	1	0.88	80.2 %	60.97 %	70.59 %	89.52 %	84.61 %
39	1	0.9	79.94 %	60.73 %	70.33 %	89.51 %	84.45 %

Table 4.4: Best results obtained with the KNN-based predictor on a focal plane with 487 astrobots. It is possible to notice that the results are slightly lower with respect to the ones of table 4.2. The main reason behind that is the large presence of astrobots in a full neighborhood configuration.

Fig. 4.18 shows the trend of precision, recall and F1 score for differents values of K. In this case the precision does not rise above 90%, but still has satisfactory values. Finally, in fig. 4.19 are reported the two ROC curves for a focal plane with 116 astrobots and a focal plane with 487 astrobots.

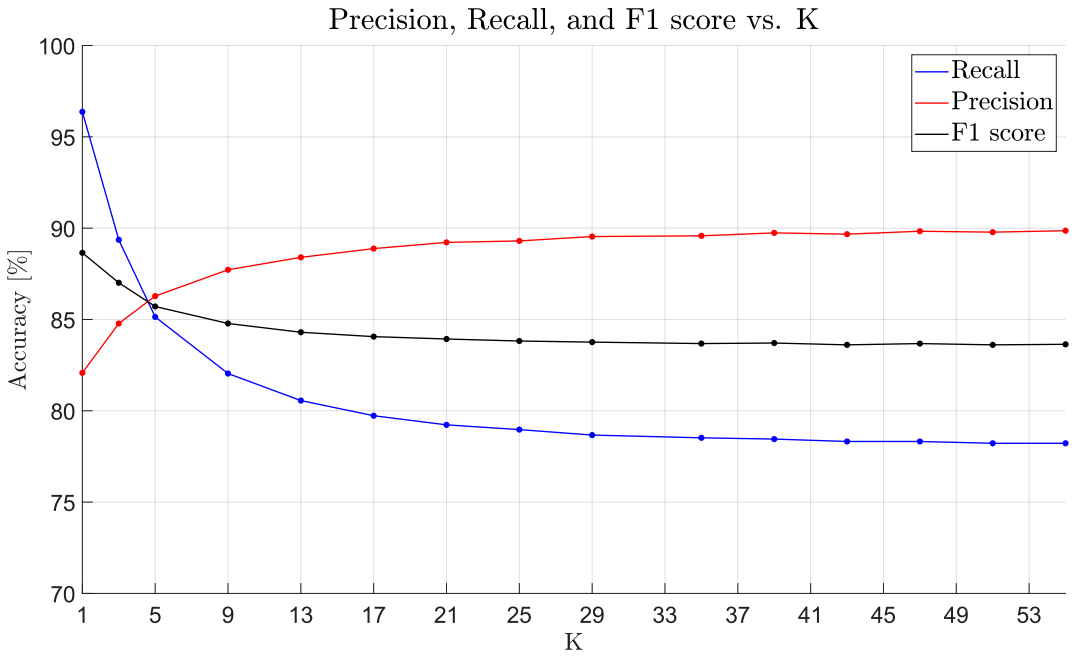


Figure 4.18: *Precision, Recall* and *F1 score* plotted for differents values of K. Even in this case the trends are stable for K greater than 21. Although the precision has a slightly less values with respect to fig. 4.12, the trade-off between precision and recall represented by the F1 score has more or less the same values of the case with a focal plane with 116 astrobots.

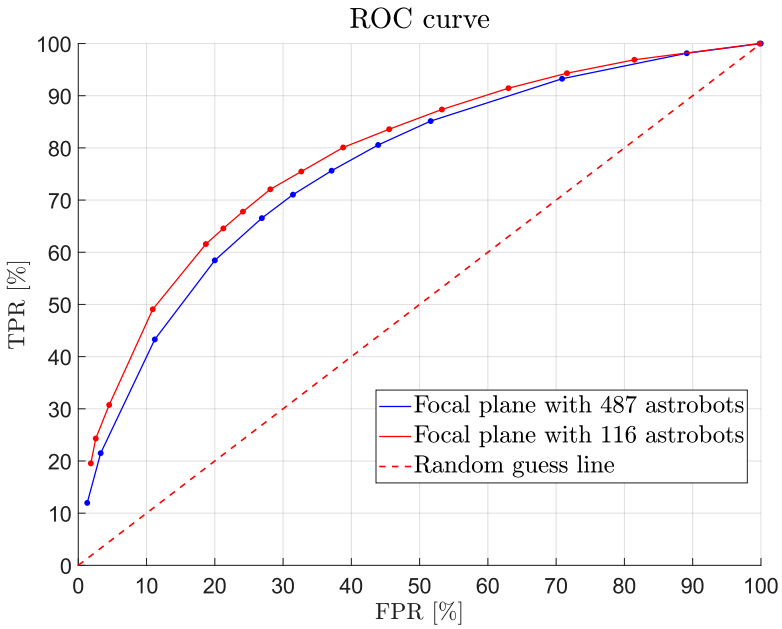


Figure 4.19: ROC curves for a focal plane with 116 astrobots and a focal plane with 487 astrobots. The curves are built using differents values of the coefficients α and β. In order to compare the two curves, in both cases K was kept fixed to 13. The fact that the blue curve is below the red one is due to the fact that in the focal plane with 487 astrobots we have a larger percentage of astrobots in a full neighborhood configuration, for whom the prediction is more critical.

4.4.2 SVM results

As said in section 2.2, in this case we are building a SVM model for each astrobot in the focal plane. Together with the training phase, the testing phase is performed using data samples of the single astrobots.
Suppose we have a number k of iterations of the cross validation process. At each iteration we build a SVM model for a single astrobot and, using the test dataset, we compute the number of true positives TP_i, true negatives TN_i, false positives FP_i and false negatives FN_i, with $i \in \{1,2,\ldots,k\}$.
Let's now indicate with TP_π, TN_π, FP_π and FN_π the average over the number of iterations k of the true positives, true negatives, false positives and false negatives respectively for a single astrobots, computed as follows:

$$TP_\pi = \frac{\sum_{i=1}^{k} TP_i}{k} \quad TN_\pi = \frac{\sum_{i=1}^{k} TN_i}{k} \quad FP_\pi = \frac{\sum_{i=1}^{k} FP_i}{k} \quad FN_\pi = \frac{\sum_{i=1}^{k} FN_\pi}{k} \tag{4.11}$$

These values represent the final number of true positives, true negatives, false positives and false negatives of a single astrobot. We repeat this procedure for each astrobot in the focal plane. At the end, in order to compute the total result for the entire focal plane we take the average of the results of the single astrobots over the total number of astrobots.
If we indicate with R the total number of astrobots, the final results are then:

$$TP = \frac{\sum_{\pi=1}^{R} TP_\pi}{R} \quad TN = \frac{\sum_{\pi=1}^{R} TN_\pi}{R} \quad FP = \frac{\sum_{\pi=1}^{R} FP_\pi}{R} \quad FN = \frac{\sum_{i=1}^{k} FN_\pi}{R} \tag{4.12}$$

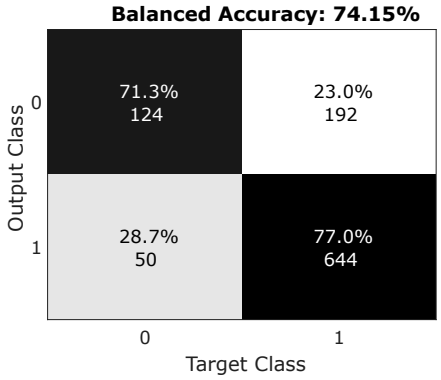
All the metrics described in sec. 4.3 are computed using the values TP, TN, FP and FN.

In a focal plane there are different types of astrobots (those with six neighbors, those with five neighbours, those with four neighbours, etc). For each type of astrobot we have to find good values of the hyperparamenters C, σ and w_1. Indeed, as stated in eq. 2.17 the dimensions of a single data vector change depending on the number of neighbours of the astrobot. As a result, the value of the hyperparameters depend on the number of neighbours of the astrobot.
Since the kernel function is the same for all the SVM models, in theory we have three hyperparameters to tune for each type of astrobot's neighborhood, for a total of 3×4 hyperparameters. However a good value of the missclassification penalty C has been found manually and it has been kept the same for all the types of astrobot. Therefore the total number of hyperparameters to tune is 2×4 in this case: for each neighborhood we have to tune the kernel size σ and the class weight w_1.
Good values of these two hyperparameters have been found performing a grid search for each type of neighborhood. The kernel function is a gaussian kernel function, while the value of C is equal to 1.
For each experiment reported the values of the hyperparameters are reported in a table.

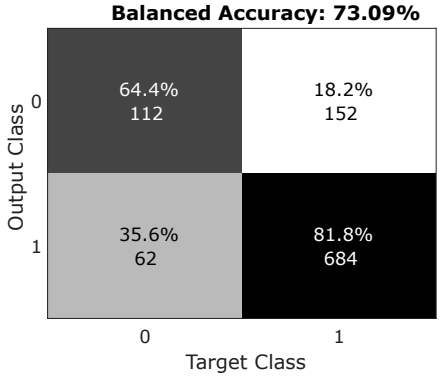
Focal plane with 116 astrobots

Also in this case, the number of simulated configurations is of 10100. As a result, each astrobot has a dataset of 10100 samples, 10% of which are used as test set, while the remaining 90% is used as training

set to build the model. It is importanto to underline that thanks to the k-folds cross validation, each data sample is used both as a training data and as a test data (see sec. 2.2.3).
The first results obtained using the best hyperparameters of the grid search are reported in fig. 4.20.



Type of neighborhood	σ	w_1
Astrobot with 6 neighbours	0.96	0.277
Astrobot with 5 neighbours	1.03	0.216
Astrobot with 4 neighbours	0.965	0.167
Astrobot with 3 neighbours	1.26	0.167



Type of neighborhood	σ	w_1
Astrobot with 6 neighbours	0.82	0.307
Astrobot with 5 neighbours	0.86	0.256
Astrobot with 4 neighbours	0.82	0.207
Astrobot with 3 neighbours	0.98	0.207

Figure 4.20: Best results obtained using the hyperparameters σ and w_1 found thanks to a grid search. On the right side are reported the hyperparameters of the prediction, while on the left side are reported the confusion matrices of the predictions made using the corresponding hyperparameters. In the first case the aim was to maximize the balanced accuracy while keeping the TPR above 75 %. In the second case the TPR has increased at the expense of the balanced accuracy, which has a lower value.

In fig. 4.20 are shown the results for the prediction on the entire focal plane. But even in this case, as was also done for the KNN algorithm, it is important to see how the results of the accuracy are distributed according to the type of the astrobot's neighborhood. In fig. 4.21 are reported the results of the neighborhood analysis computed using the hyperparameters of the second table of fig. 4.20.

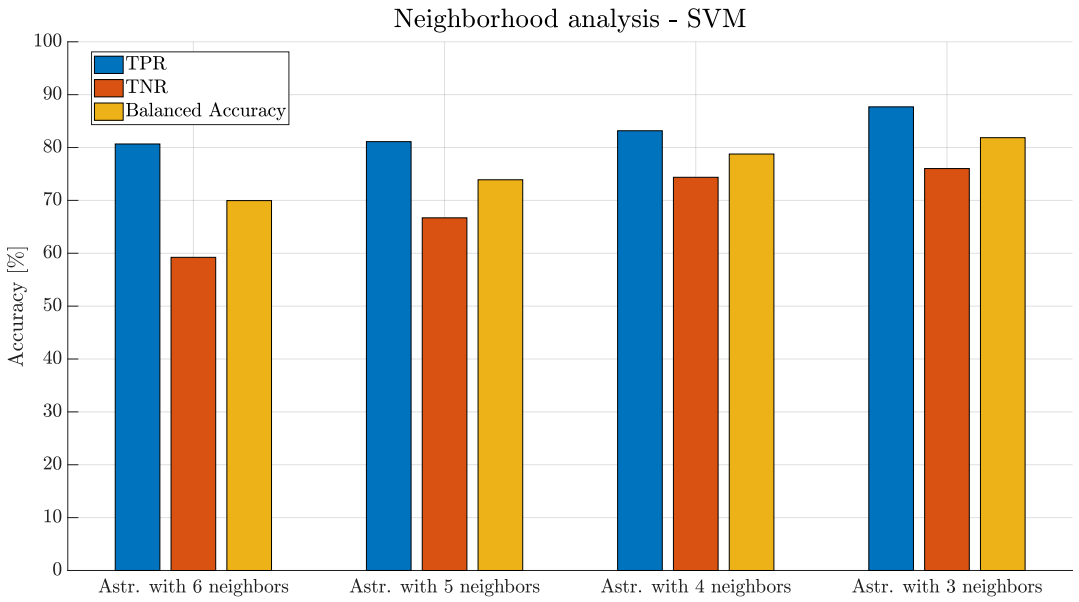


Figure 4.21: Neighborhood analysis for the SVM prediction using the hyperparameters reported in the second table of fig. 4.20. Even in this case it is possible to observe that the worst condition for prediction is for the astrobots with six neighbours. The prediction performance improves by decreasing the number of neighbours of the astrobots.

From fig. 4.21 it is possible to notice that the TPR is larger with respect to the TNR in each type of neighborhood. Indeed in this case values of the hyperparameters were chosen which guaranteed a higher percentage of the TPR with respect to the percentage of the TNR. It is possible change these percentages by varying the class weight w_1 of the majority class. In particular, if we decrease w_1 we are lowering the weight of the positives, and consequently the TPR will decrease.
In fig. 4.22 it is possible to visualize the effect of decreasing the weights w_1 of the positive class.

Type of neighborhood	σ	w_1
Astrobot with 6 neighbours	0.82	0.227
Astrobot with 5 neighbours	0.86	0.206
Astrobot with 4 neighbours	0.82	0.147
Astrobot with 3 neighbours	0.96	0.147

Table 4.5

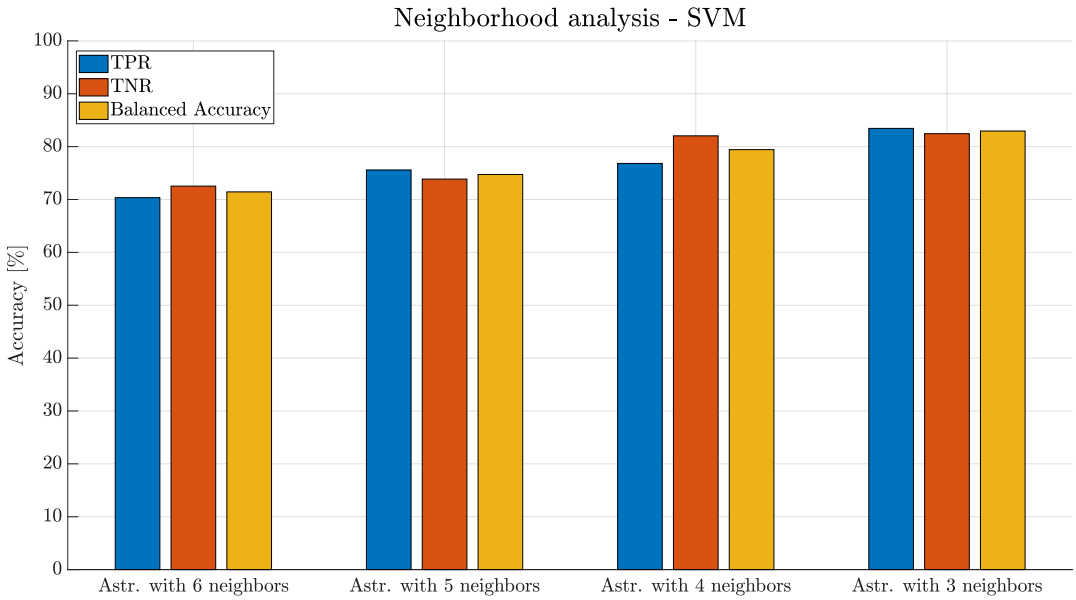


Figure 4.22: Neighborhood analysis for the SVM prediction using the hyperparameters reported in table 4.5. It is possible to notice that the TNR for each type of astrobot has increased with respect to the fig. 4.21. This because the class weights w_1 of the table 4.5 are smaller with respect to the ones of the second table of fig. 4.20. As a result the predictor classifies in a better manner the negatives and in a worse manner the positives.

To visualize the effect of decreasing the weights w_1 on the TPR, TNR and *Balanced Accuracy* indices for the entire focal plane, it has been used the weighted average of the weights w_1 between the different types of astrobots.
For example, let's indicate with $w1_6$, $w1_5$, $w1_4$ and $w1_3$ the weights w_1 used respectively for the astrobots with six, five, four and three neighbors. Let's indicate now with n_6, n_5, n_4 and n_3 respectively the total number of astrobots in the focal plane with six, five, four and three neighbours.
The weighted average of the weights $\bar{w_1}$ between the different types of astrobot is:

$$\bar{w_1} = \frac{w1_6 \cdot n_6 + w1_5 \cdot n_5 + w1_4 \cdot n_4 + w1_3 \cdot n_3}{n_6 + n_5 + n_4 + n_3} \tag{4.13}$$

The equation 4.13 can be generalized as follows:

$$\bar{w_1} = \frac{\sum_{i=1}^{6} w1_i \cdot n_i}{\sum_{i=1}^{6} n_i} \tag{4.14}$$

Where $w1_i$ indicates the weight w_1 for the astrobots with a number i of neighbours, while n_i indicates the total number of astrobots with a number i of neighbours.
In fig. 4.23 are reported the the TPR, TNR and *Balanced accuracy* when varying $\bar{w_1}$ with a fixed kernel size for each astrobot.

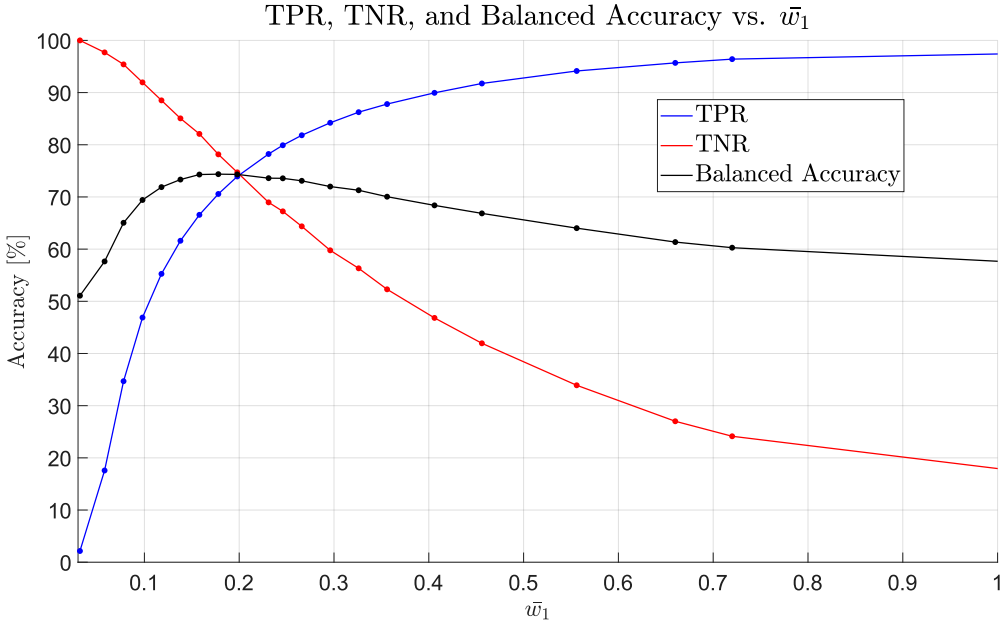


Figure 4.23: TPR, TNR and *Balanced accuracy* when varying the weighted average $\bar{w_1}$ of the class weights w_1 of the different types of astrobots. The kernel size is fixed for each astrobot. It is evident that the increase of the positive class weights causes an increase in the TPR and a decrease in the TNR. This is reasonable, because if we increase $\bar{w_1}$ we are giving greater weight to the class of positives and consequently the predictor classifies positives better. Also in this case, as for the KNN (fig. 4.9), the balanced accuracy reaches a maximum when the TPR and TNR have more or less the same value.

Always using the average positive class weight $\bar{w_1}$ as variable for the performances analysis, it possible to plot the precision, the recall and the F1 score as function of $\bar{w_1}$.

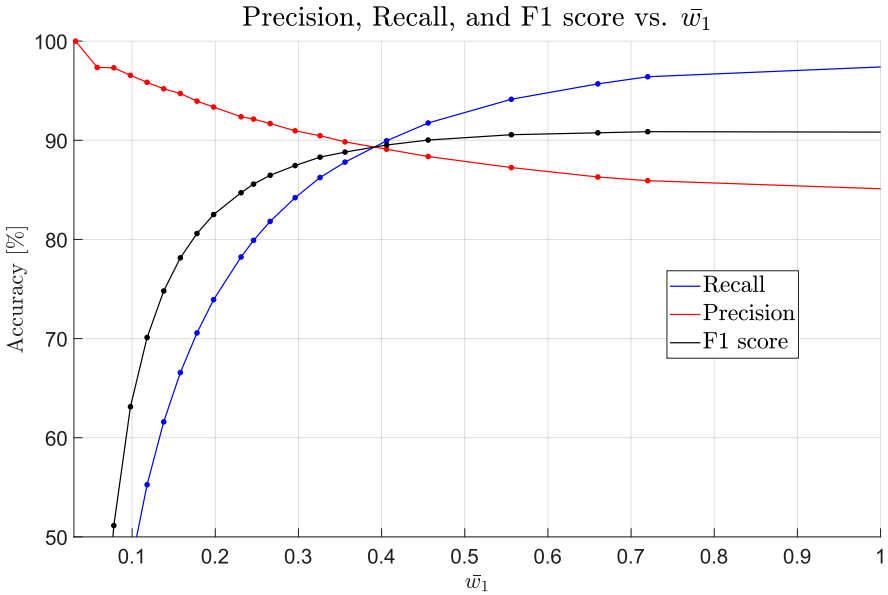


Figure 4.24: *Recall, Precision* and *F1 score* when varying the average class weight $\bar{w}_1$. The kernel size σ are fixed for each type of astrobot. According to figure 4.23, the TPR (here it is the recall) increase with $\bar{w}_1$. However, in addition to the increase of true positives, a larger $\bar{w}_1$ causes also an increase of the false positives. As a result, according to eq. 4.6 there is a decrease of the precision. However we cannot look only at the precision when analysing the predictor's ability in classifying positives. Indeed high values of the precision correspond to low values of the recall (the TPR). The F1 score then represent a good index for the trade-off between precision and recall. If the F1 score decreases too much, we can be sure that one index between precision and recall has a very low value, and the performance of the predictor are not good.

From fig. 4.24 it is possible to notice that the best F1 score is reached for high values of $\bar{w}_1$. However from fig. 4.23 we can observe that for high values of the average class weight $\bar{w}_1$ the balanced accuracy has low values (below 70%). Then a possible selection criteria could be to maximize the F1 score keeping the balanced accuracy above 70%.

It is possible to perform an analysis similar to the one made for weights w_1 but this time using the kernel size.

Let's denote with $\bar{\sigma}$ the weighted average of the kernel sizes of the different types of astrobots:

$$\bar{\sigma} = \frac{\sum_{i=1}^{6} \sigma_i \cdot n_i}{\sum_{i=1}^{6} n_i} \tag{4.15}$$

Where σ_i indicates the kernel size σ for the astrobots with a number i of neighbours, while n_i indicates the total number of astrobots with a number i of neighbours.
In fig. 4.25 are reported the the TPR, TNR and *Balanced accuracy* when varying $\bar{\sigma}$ and keeping fixed the class weights w_1 of each astrobot.

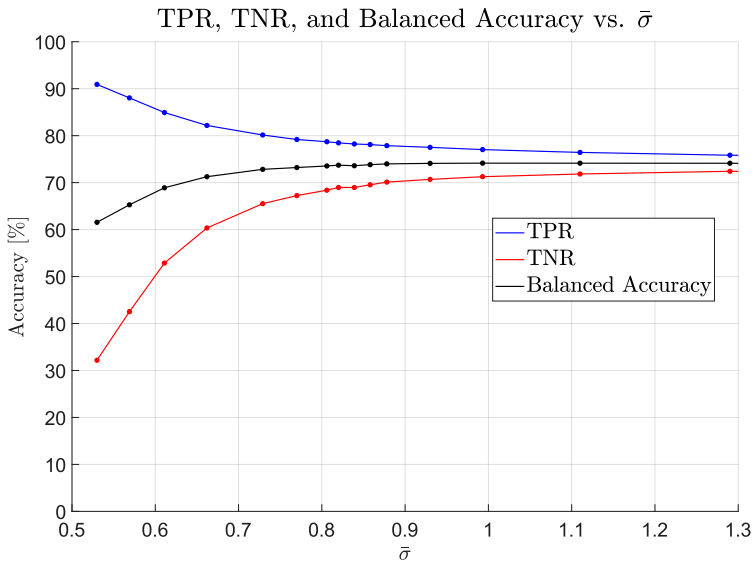


Figure 4.25: TPR, TNR and *Balanced accuracy* when varying the weighted average $\bar{\sigma}$ of the kernel size σ of the different types of astrobots. The class weights w_1 are fixed. An increase of the kernel size leads to a better classification of the negatives and a worse classification of the positives. On the other hand, a decrease of the kernel size causes a better classification of the positives and a worse classification of the negatives. This can be explained by the fact that when the kernel size is too small or too large, the gaussian function that maps the distance between the vectors in the higher dimensional space is not able to make the two classes sufficiently separable from each other.

In table 4.6 are reported the best results with the SVM predictor on this focal plane and we can look at the ROC curve in fig. 4.26 to understand if our predictor's behaviour is sufficiently far from a random classifier. In this case the hyperparameter which varies at each prediction simulation is the average class weight $\bar{w_1}$. It is also reported the ROC curve for the KNN-based predictor for a focal plane with 116 astrobots, in order to compare the results.

Type of neighborhood	σ	w_1	**TPR**	**TNR**	**Bal. Acc.**	**Precision**	**F1 score**
Astr. with 6 neighbours	0.96	0.277	77%	71.3%	74.15%	92.79%	84.18%
Astr. with 5 neighbours	1.03	0.216					
Astr. with 4 neighbours	0.965	0.167					
Astr. with 3 neighbours	1.26	0.167					
Astr. with 6 neighbours	0.82	0.307	81.8%	64.4%	73.1%	91.69%	86.47%
Astr. with 5 neighbours	0.86	0.256					
Astr. with 4 neighbours	0.82	0.207					
Astr. with 3 neighbours	0.98	0.207					

Table 4.6: Best results for the SVM-based predictor on a focal plane with 116 astrobots. In the first case the aim was to maximize the *Balanced Accuracy* while keeping the TPR above 75%. In the second case the TPR has increased to more than 80% and the *balanced accuracy* has still an high values.

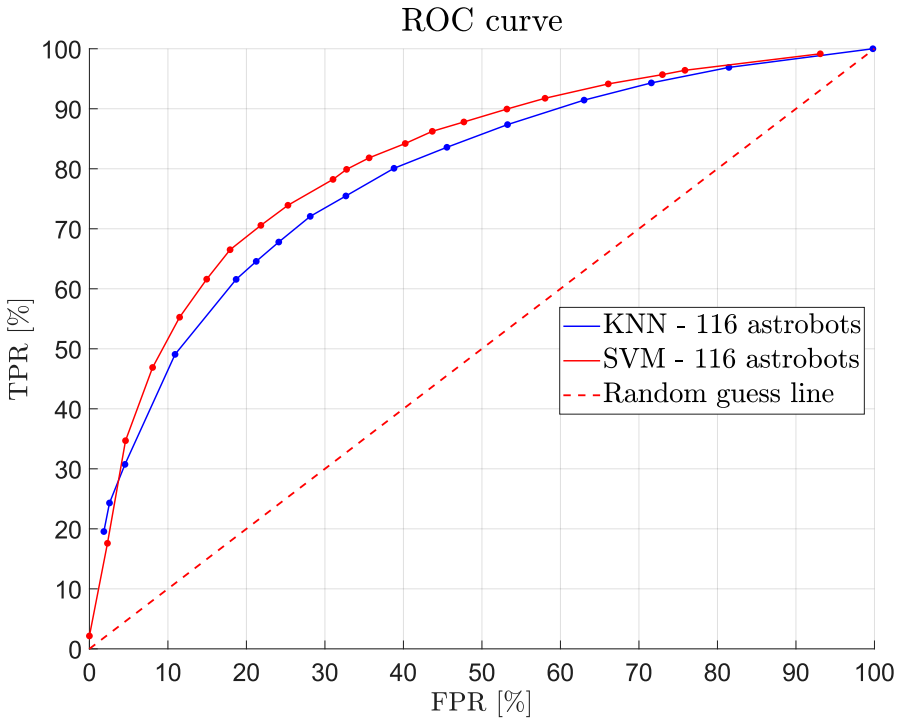
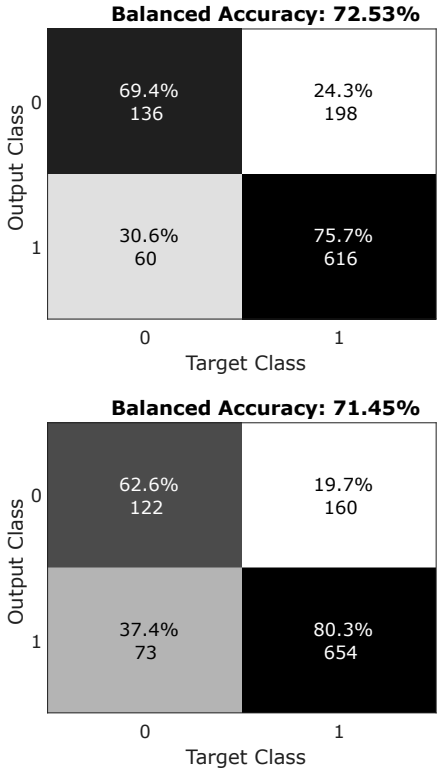


Figure 4.26: ROC curves for the KNN-based predictor and SVM-based predictor on the same focal plane with 116 astrobots. It is possible to observe that the performances of the SVM-based predictor are slightly better then the ones of the KNN-based predictor.

Focal plane with 487 astrobots

Each astrobot has a dataset of 10100 samples, and also in this case the percentage of the dataset used as test dataset is of 10%. It is necessary to underline again the relevant presence of the astrobots in a full neighborhood configuration, therefore a slight decrease in terms of performance is expected with respect to the focal plane with 116 astrobots.
In fig. 4.27 are reported two relevant results obtained using the best hyperparameters of the grid search.



Type of neighborhood	σ	w_1
Astrobot with 6 neighbours	0.86	0.277
Astrobot with 5 neighbours	0.9	0.216
Astrobot with 4 neighbours	0.86	0.167
Astrobot with 3 neighbours	1.47	0.167
Astrobot with 2 neighbours	0.84	0.179

Type of neighborhood	σ	w_1
Astrobot with 6 neighbours	0.82	0.317
Astrobot with 5 neighbours	0.86	0.256
Astrobot with 4 neighbours	0.82	0.207
Astrobot with 3 neighbours	0.98	0.207
Astrobot with 2 neighbours	0.8	0.219

Figure 4.27: Best results obtained using the hyperparameters σ and w_1 found thanks to a grid search. On the right side are reported the hyperparameters of the prediction, while on the left side are reported the confusion matrices of the predictions made using the corresponding hyperparameters. In the first case the aim was to maximize the balanced accuracy while keeping the TPR above 75%. In the second case the TPR has reached a value of 80% at the expense of the balanced accuracy, because the TNR has decreased.

In fig. 4.27 it is possible to observe that the results obtained have a lower value with respect to the ones of fig. 4.20. The reason for this lies in the larger presence of astrobots in a full neighborhood configuration, for whom prediction is more critical.

In fig. 4.28 is reported the neighborhood analysis using the hyperparameters of the second table of fig. 4.27. We can observe that the values of TPR, TNR and balanced accuracy are very close to the ones of fig. 4.21. Indeed many of the hyperparameters chosen in the two cases are the same. This is a further evidence that the performances of the single astrobots do not depend on the size of the focal plane. This feature can be used to make predictions on much larger focal planes without generating a new dataset for the specific focal plane under analysis.

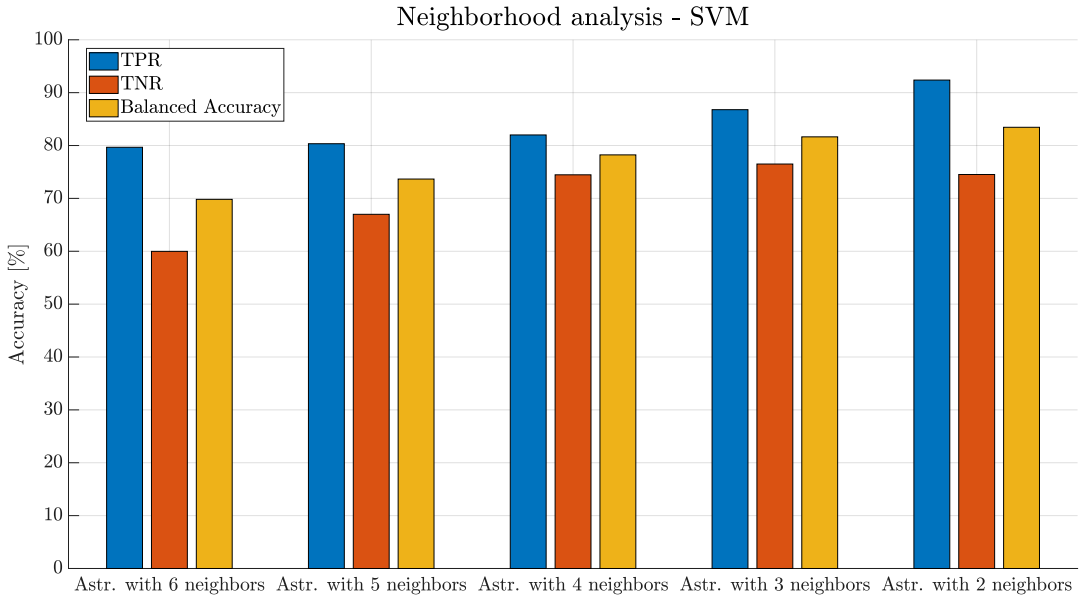
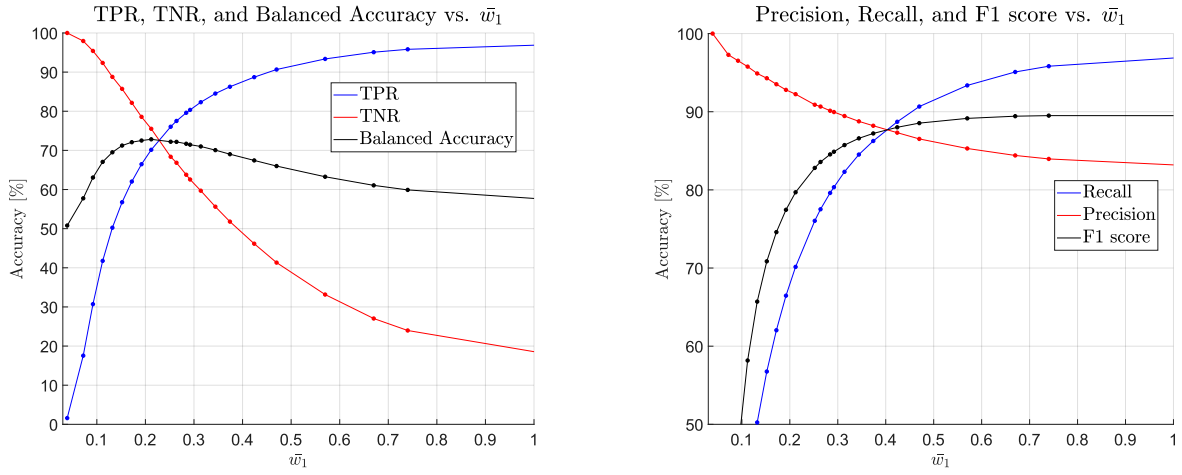


Figure 4.28: Neighborhood analysis for the SVM prediction on a focal plane with 487 astrobots using the hyperparameters reported in the second table of fig. 4.27.



(a) (b)

Figure 4.29: Performances of the predictor on a focal plane with 487 astrobots when varying the average class weight $\bar{w}_1$. In fig. (a) are reported the trends of TPR, TNR and *Balanced accuracy*, while in fig. (b) are reported the trends of the precision, the recall and the F1 score. It is possible to observe that the trends are similar to the ones obtained with a focal plane with 116 astrobots.

The average class weight $\bar{w_1}$ has been computed using again eq. 4.14. In fig. 4.30 is reported the trends of the TPR, TNR and Balanced accuracy when varying the average kernel size computed using eq. 4.15.

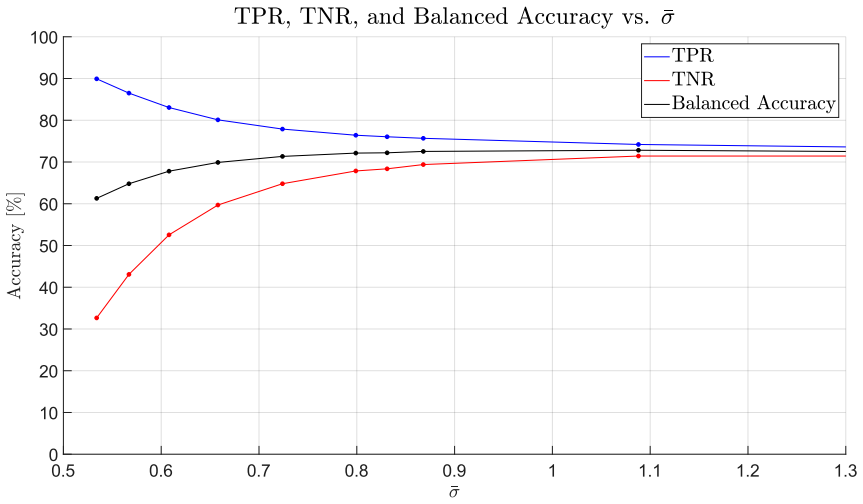


Figure 4.30: TPR, TNR and *Balanced accuracy* when varying the weighted average $\bar{\sigma}$ of the kernel size σ of the different types of astrobots. The trends are very similar to the case with a focal plane with 116 astrobots, and the same considerations hold.

The best results of the predictor on the focal plane with 487 astrobots are reported on the table 4.7. The ROC curve in fig. 4.31 allows to visualize the improvement with respect to the KNN-based predictor on the same focal plane.

Type of neighborhood	σ	w_1	**TPR**	**TNR**	**Bal. Accuracy**	**Precision**	**F1 score**
Astr. with 6 neighbours	0.86	0.277	75.7%	69.4%	72.5%	91.12%	82.68%
Astr. with 5 neighbours	0.9	0.216					
Astr. with 4 neighbours	0.86	0.167					
Astr. with 3 neighbours	1.47	0.167					
Astr. with 2 neighbours	0.84	0.179					
Astr. with 6 neighbours	0.82	0.317	80.3%	62.6%	71.45%	89.96%	84.88%
Astr. with 5 neighbours	0.86	0.256					
Astr. with 4 neighbours	0.82	0.207					
Astr. with 3 neighbours	0.98	0.207					
Astr. with 2 neighbours	0.8	0.219					

Table 4.7: Best results of the SVM-based predictor on a focal plane with 487 astrobots. As for the focal plane with 116 astrobots, in the first case the aim was to maximize the *Balanced Accuracy* while keeping the TPR above 75%. In the second case we have an increase of the TPR to above 80% while mantaining a good value for the *Balanced Accuracy*.

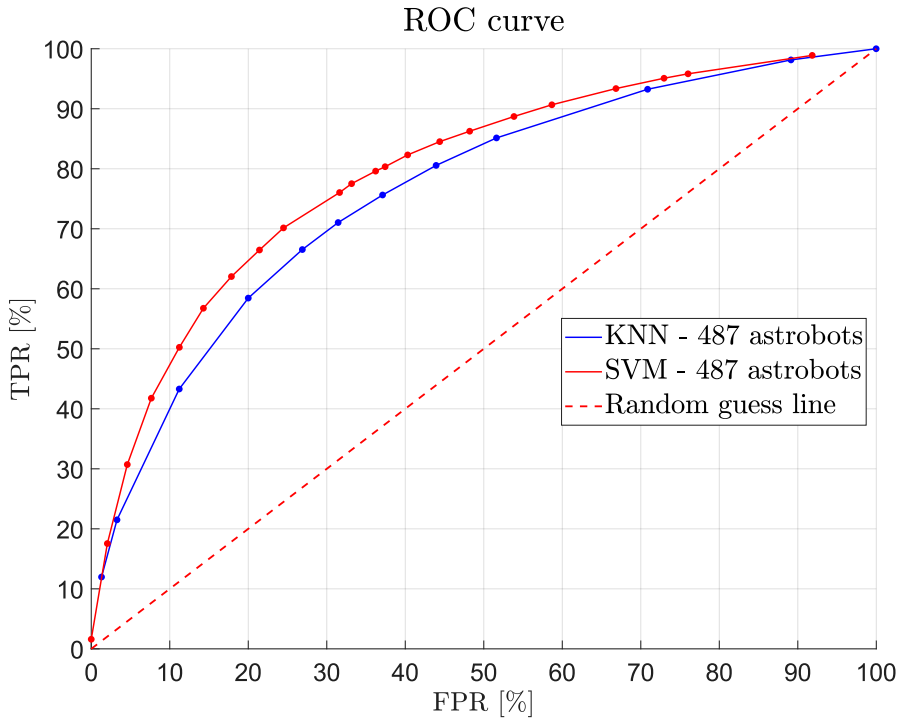


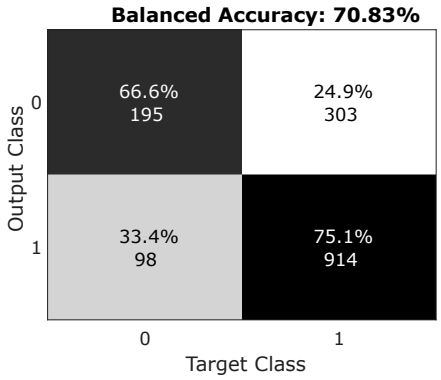
Figure 4.31: ROC curves for the KNN-based predictor and SVM-based predictor on the same focal plane with 487 astrobots. Even in this case the SVM predictor shows an improvement of performances with respect to the KNN-based predictor.

4.5 Parity-variable prediction results

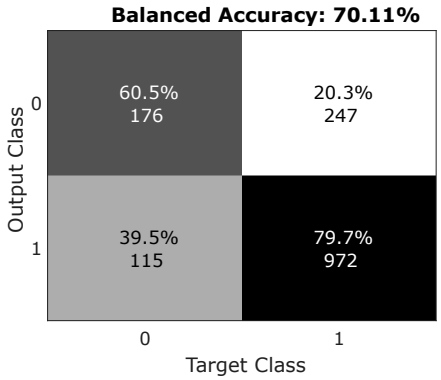
In this section are reported the results of the parity-variable prediction using the SVM-based algorithm.

4.5.1 SVM results

The performances of the parity-variable prediction algorithm are evaluated on the focal plane with 487 astrobots, since it represents the most critical condition in terms of prediction results. given the large presence of astrobots in a full neighborhood configuration. In this case the dataset includes 15100 data samples, 5000 more than in the case of parity-based prediction. The choice of having a number of data samples greater with respect to the parity-based case is related to the increase of the possible targets-parity combinations given by the introduction of the parity itself.
Also in this case the number of iterations k of the cross-validation process has been set to 10. The number of true positives, true negatives, false positives and false negatives used inside the metrics is the same described in sec. 4.4.2. The best results are reported in fig. 4.32.



Type of neighborhood	σ	w_1
Astrobot with 6 neighbours	1.55	0.29
Astrobot with 5 neighbours	1.55	0.25
Astrobot with 4 neighbours	1.17	0.2
Astrobot with 3 neighbours	0.99	0.14
Astrobot with 2 neighbours	0.99	0.11



Type of neighborhood	σ	w_1
Astrobot with 6 neighbours	1.42	0.33
Astrobot with 5 neighbours	1.42	0.29
Astrobot with 4 neighbours	1.05	0.24
Astrobot with 3 neighbours	1.09	0.18
Astrobot with 2 neighbours	1.09	0.15

Figure 4.32: Best results for the parity-variable prediction obtained using the hyperparameters σ and w_1 found thanks to a grid search. On the right side are reported the hyperparameters of the prediction, while on the left side are reported the confusion matrices of the predictions made using the corresponding hyperparameters. In the first case the aim was to maximize the balanced accuracy while keeping the TPR above 75%. In the second case we tried to maximize the TPR while keeping the balanced accuracy above 70 %

In fig. 4.33 is reported the neighborhood analysis for a prediction carried out using the hyperparameters of the first table of fig. 4.32. It is possible to observe that the behaviour of the predictor on the different neighborhoods is close to the one obtained with a fixed parity. Also the analysis performed by varying the average class weight $\bar{w_1}$ confirms that the trends obtained are similar to the parity-based case. The best results are are reported in table 4.8 while in fig. 4.35 it is possible to observe the trends of the TPR, TNR and balanced accuracy when varying the average kernel size $\bar{\sigma}$.

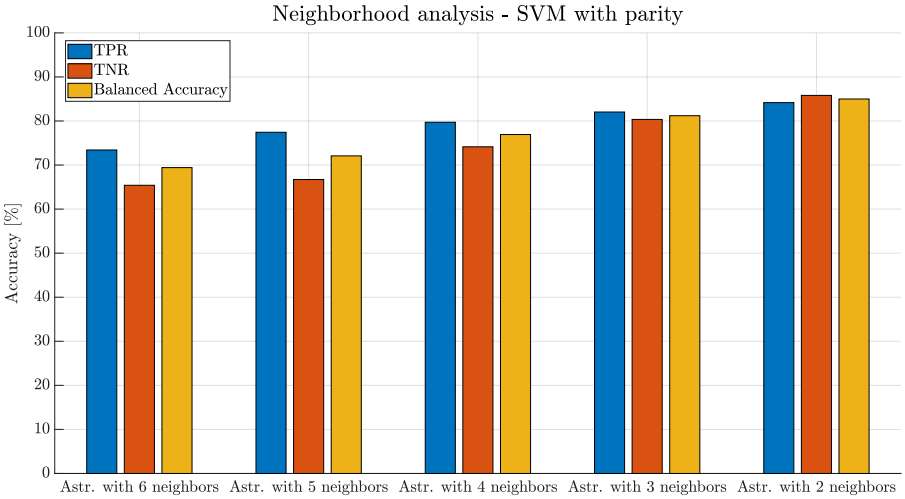
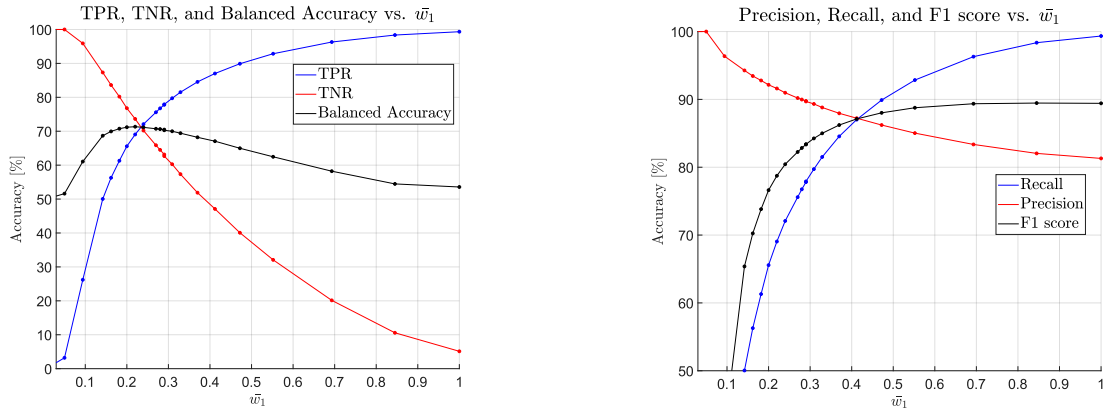


Figure 4.33: Neighborhood analysis for the SVM parity-variable prediction on a focal plane with 487 astrobots using the hyperparameters reported in the first table of fig. 4.32.



(a) (b)

Figure 4.34: Performances of the SVM-based predictor when the parity information is embdedded inside the data samples and when varying the average class weight $\bar{w_1}$. In fig. (a) are reported the trends of TPR, TNR and *Balanced accuracy*, while in fig. (b) are reported the trends of the precision, the recall and the F1 score. Both the trends are very similar to the parity-based case reported in fig. 4.29.

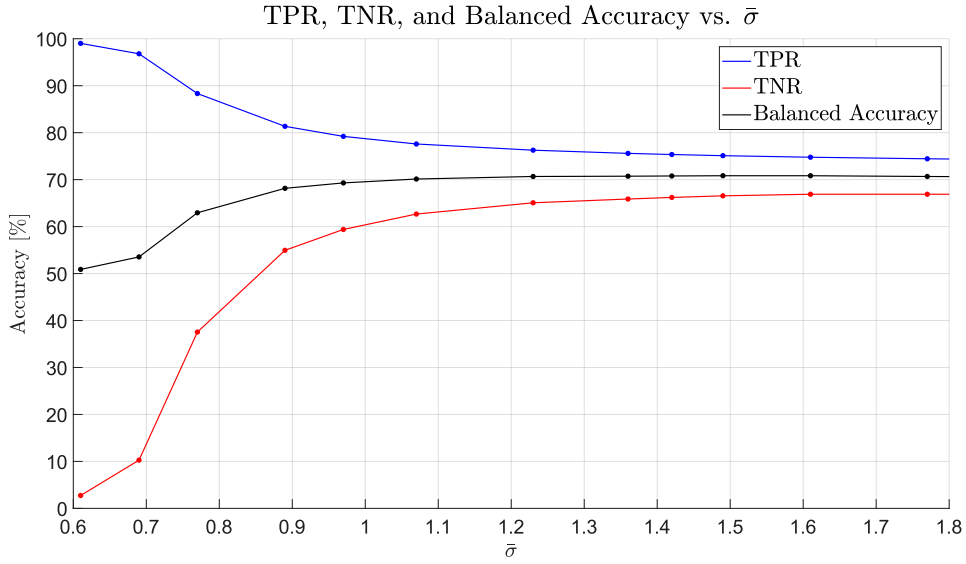


Figure 4.35: TPR, TNR and *Balanced accuracy* when varying the weighted average $\bar{\sigma}$ of the kernel size σ of the different types of astrobots. The trends are very similar to the parity-based case, the only difference is a general increase of the average kernel size $\bar{\sigma}$, given by the introduction of the parity inside the data samples. As a matter of fact the vectors are now on average more distant from each others, and consequently the kernel size has to be larger with respect to the parity-based case.

Type of neighborhood	σ	w_1	**TPR**	**TNR**	**Bal. Accuracy**	**Precision**	**F1 score**
Astr. with 6 neighbours	1.55	0.29					
Astr. with 5 neighbours	1.55	0.25					
Astr. with 4 neighbours	1.17	0.2	75.1%	66.6%	70.8%	90.3%	82%
Astr. with 3 neighbours	0.99	0.14					
Astr. with 2 neighbours	0.99	0.11					
Astr. with 6 neighbours	1.42	0.33					
Astr. with 5 neighbours	1.42	0.29					
Astr. with 4 neighbours	1.05	0.24	79.7%	60.5%	70.1%	89.4%	84.3%
Astr. with 3 neighbours	1.09	0.18					
Astr. with 2 neighbours	1.09	0.15					

Table 4.8: Best results of the SVM-based predictor on a focal plane with 487 astrobots in the case of parity-variable prediction. In the first case the aim was to maximize the *balanced accuracy*, in the second case the aim was to maximize the TPR

In fig. 4.36 is reported the comparison between the ROC curves of the parity-based predictors and the parity-variable predictor.

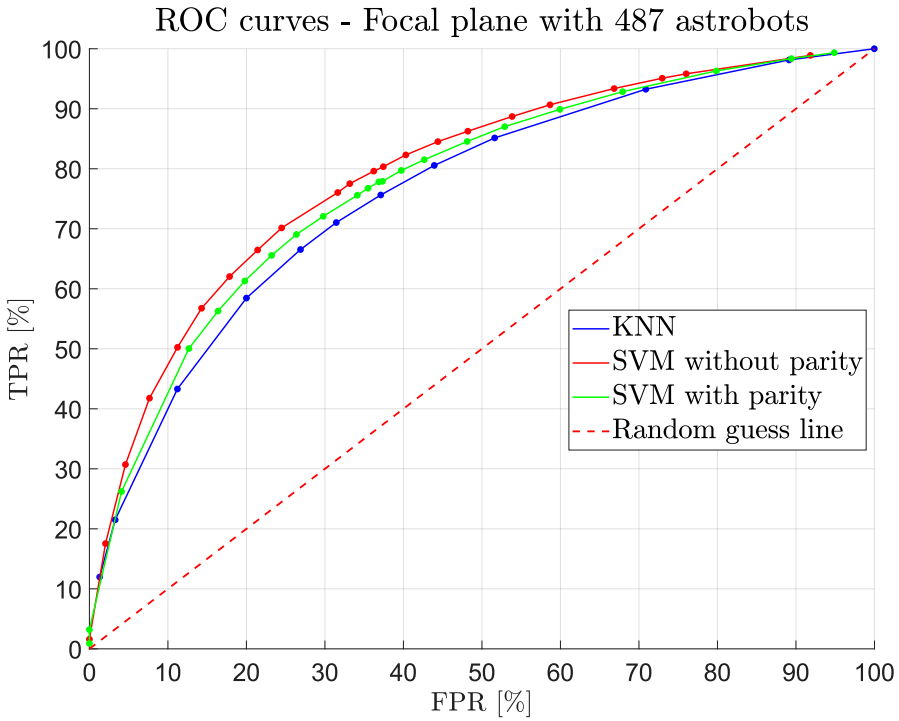
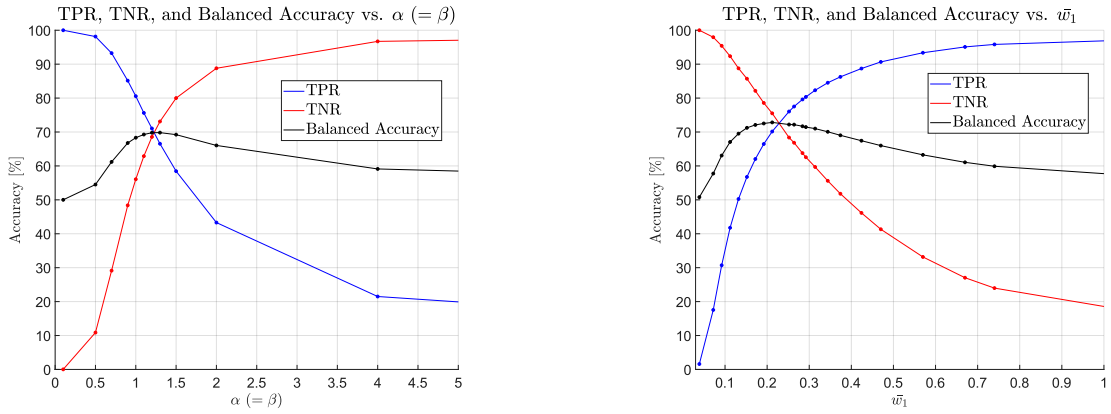


Figure 4.36: Comparison between the two parity-based predictors (KNN and SVM) and the SVM parity-variable predictor on a focal plane with 487 astrobots. It is possible to see that the performances of the parity-variable predictor are slightly worse with respect to the parity-based case, but they are still better than the KNN-based predictor.

Chapter 5

Discussion

The results obtained shows that is possible to reach 80% of accuracy in the prediction of the astrobots which converge to their target position. However, in both the prediction algorithm developed, a good prediction of the positives is attained at the expense of an accurate prediction of the negatives, which is between 60% and 70%. The trade-off between a correct prediction of the positives and the correct prediction of the negatives is regulated in both algorithm by the class weight. Although the class weight acts differently in the two algorithms, the effects of its variation are similar in both cases, as shown in fig. 5.1.



(a) (b)

Figure 5.1: Trends of the TPR, TNR and *balanced accuracy* for the two predictors when varying the class weight in a focal plane with 487 astrobots. The trends of the TPR and TNR are opposite in the two cases because in the KNN (a) the parameters α and β act on the weight of the negative class, while in the SVM (b) the parameters $\bar{w}_1$ acts on the weight of the positive class.

The value of the balanced accuracy is maximum when the TPR and the TNR have more or less the same value. This means that in order to increase the balanced accuracy we must allow a decrease of the TPR below the value of 80%.

The core idea behind the two prediction algorithms is to carry out a local analysis, applying the algorithms not to the entire focal plane but rather on small subset which include a maximum number of astrobots equal to 7. This approach allows to get more reliable results than an analysis that includes the entire focal plane, since the number of possible target and parity configurations grows exponentially with the number of astrobots. As a result, if one wants to perform a global prediction on the whole focal plane, the number of data samples needed would be enormous. Contrarily a local analysis allows to have valid results with relatively limited amount of data.
Another advantage of the local analysis is that it allows performance projections on very large focal planes. In fact, knowing the performances of the single types of neighborhood and knowing the number of different types of neighborhood in the focal plane, it is possible to get an estimate of the predictor's performances. For example, from figure 4.28 it is possible to observe what are the performances of the SVM-based predictor for the single types of neighborhood in a focal plane with 487 astrobots. Knowing these results it is possible to estimate the performances of the predictor on a larger focal plane by considering the neighborhood distribution in the new focal plane. In general, the larger is the focal plane, the larger is the presence of astrobots in a full neighborhood configuration. Hence it is reasonable to conjecture that the performances on very large focal planes will tend to be close to the prediction results for astrobots in a full neighborhood configuration.

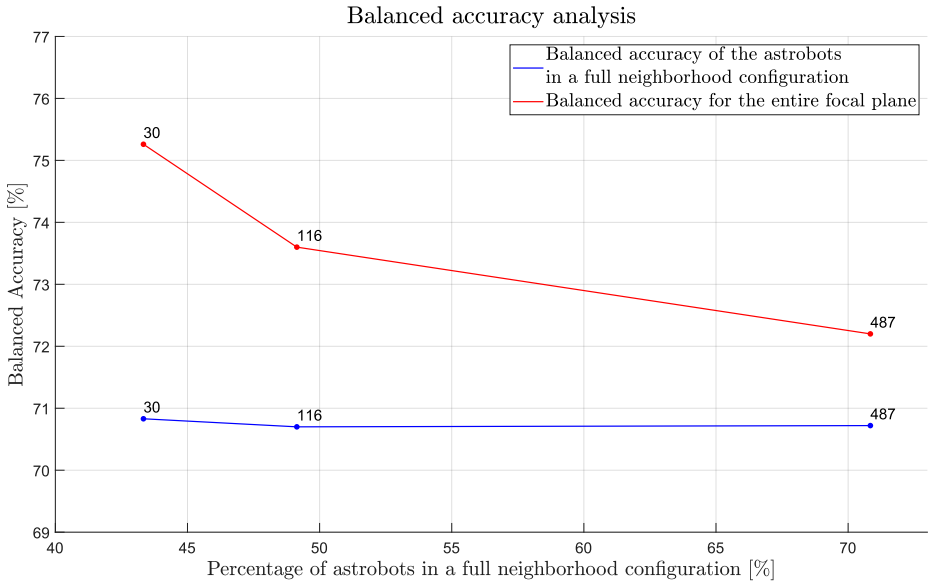


Figure 5.2: Trends of the balanced accuracy for the astrobots in a full neighborhood configuration and for the entire focal plane, depending on the percentage of astrobots in a full neighborhood configuration. Near the points on the graphs are indicated the total number of astrobots in the focal plane. It is possible to observe that the value of the balanced accuracy for the astrobots in a full neighborhood configuration is more or less stable across the different sizes of the focal plane, indicating that the local analysis does not depend on the size of the focal plane. On the other hand, the balanced accuracy for the entire focal plane depends on the size of the focal plane, and it is possible to observe that its value tends to be close to the balanced accuracy for the astrobots in a full neighborhood configurations when the precentage of astrobots in a full neighborhood configuration increases.

From the results obtained, it is possible to observe that the SVM-based algorithm provides slightly better results with respect to the KNN-based algorithm. The use of a kernel function indeed allows to model in a better manner the nonlinearities present in the distribution of the data. However the SVM-based algorithm requires the tuning of a number of hyperparameters greater than the KNN-based algorithm. Furthermore, the SVM-based agorithm requires a training phase to build the model using the data samples of the training dataset. This phase can be particularly time consuming for large focal planes, but once done it allows the evaluation of a test sample in a small amount of time. On the other hand, since the KNN is a memory-based algorithm, the training phase is given by the analysis of the most similar configuration in the training dataset when a test sample is evaluated. Consequently, the algorithm is slower with respect to the SVM in evaluating one test sample.
In any case, both algorithms provide the prediction for a test sample in a much shorter time window with respect to a simulation. This was one of the most important requirements of the predictor, because the decision-making process before the start of the coordination of the astrobots has to happen in a small amount of time. In table 5.1 is reported the comparison between the times needed to run a simulation and the times needed to run a prediction.

Swarm population	**Simulation time**	**Prediction time - KNN**	**Prediction time - SVM**
116	$22 \sim 25$ sec.	$4 \sim 5$ sec.	$2 \sim 3$ sec.
487	$120 \sim 140$ sec.	$20 \sim 25$ sec.	$8 \sim 12$ sec.

Table 5.1: Comparison between prediction and simulation time for a single test configuration in the case of parity-based prediction. The values depend on the computational power of the machine where the test are carried out.

Remark. The symbol $\sim$ in table 5.1 is used to denote a range comprised between the left-side argument and the right-side argument of the symbol.

The SVM algorithm allows to relax the assumption of parity-based prediction, ensuring good results even when the information of the parity is embedded inside the data samples. Therefore the SVM algorithm turns out to be the best of the two developed. In fact, although it requires more hyperparameters to tune in order to get a good prediction and has a more complicated geometric interpretation with respect to the KNN, it gurantees a better accuracy and shorter prediction times.

Chapter 6

Remarks on potential application of a convolutional neural network

One of the most used tools in the field of image classification is definitely the convolutional neural network (CNN). The CNN is a particular type of feed-forward neural network whose connectivity pattern between neurons is inspired by the visual cortex of animals. The CNN can be trained to obtained a categorical or numerical label when an input data sample is provided. In our case the label is a binary value representing if a specific astrobot in the focal plane is able to reach its target after the coordination process. In order to exploit the potential of a CNN, it is necessary to have a data structure which is suitable for this kind of network. In fact, in the case of image classification, an image can be seen as a tensor with dimensions [*height*, *width*, *channels*], where *height* and *width* represent the image dimensions in terms of pixels quantity, while the *channels* are 3 in the case of RGB scale, or just 1 in the case of greyscale images.
Therefore, as a first step for a possible application of the CNN to our classification problem, it is necessary to rearrange the data in a structure which can be provided to the network.

6.1 Reorganization of the data

An information that is not exploited in the case of the SVM algorithm is the spatial arrangement of the astrobots inside the neighborhood. Indeed in the SVM algorithm a neighborhood is representd by a vector whose elements are the targets coordinates and the parity of the astrobots in the neighborhood itself. However we don't provide the information of the spatial arrangement of the astrobots in the neighborhood. In a CNN framework, this type of information could be implicitly provided through the input data structure. In particular, a neighborhood can be represented by a matrix whose entries positions represent the location of the astrobot in the neighborhood. If we consider a neighborhood with a maximum of 7 astrobots, a possible implementation of this strategy consists in using a 3×3 matrix for each parameter of the astrobots (i.e. targets coordinates and parity), as shown in fig. 6.1.

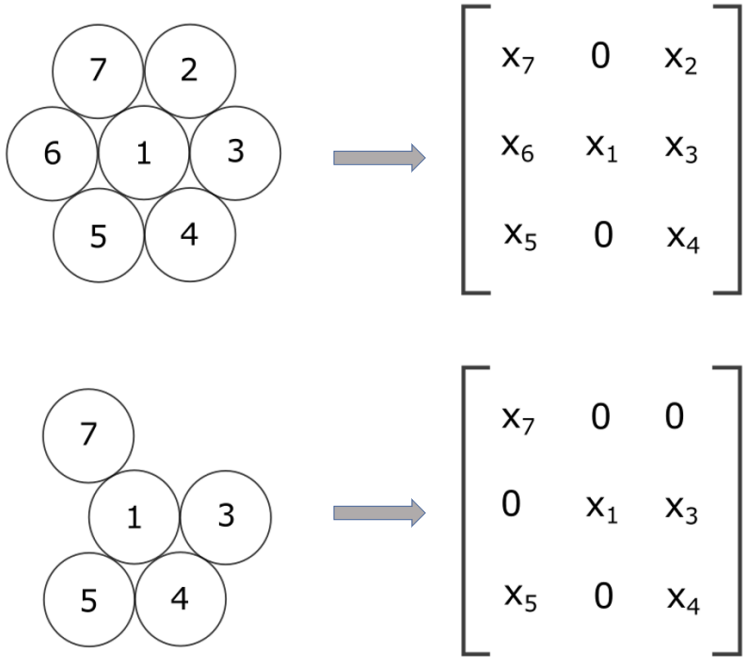


Figure 6.1: Data arrangement in the matrices according to the spatial disposition of the astrobots in the neighborhood. On the left is shown the arrangement of the astrobots in a given neighborhood, on the right is reported the corresponding matrix for the x coordinates of the astrobot. The same approach is used to build the matrices with the y coordinates and the matrices with the parity information.

Thus, for each neighborhood (and therefore for each astrobot) we build two 3×3 matrices if we consider only the targets coordinates, while if we consider also the parity we would have three 3×3 matrices. At this point the matrices are stacked along the third dimension to form a tensor of dimensions $[3, 3, 2]$ ($[3, 3, 3]$ in the parity-variable case). The first two dimensions of the tensor represent the neighborhood dimension, while the third dimension represent the number of parameters needed to describe each astrobot, and they are the targets coordinates and eventually the parity. Hence, a single data sample is represented by a tensor.

6.2 CNN structure

A convolutional layer is based on an operation called convolution. This operation allows to extract features from an image. A kernel (or filter) is applied to the image. The kernel entries multiply the entries of the image in an element-wise manner, then the products are summed together to form a new output image, as shown in fig. 6.2. The multiplication factors (also called weights) inside the filter are not fixed. They change at every iteration of the training process according to the backpropagation algorithm [22].

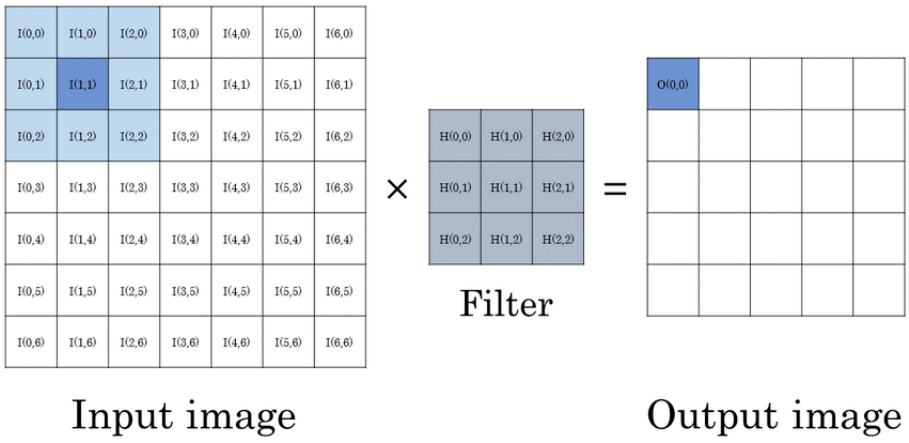


Figure 6.2: Convolution product (reprinted with permission of [23]).

The filter is applied repeatedly to the input tensor, moving gradually until completely reconstructing an output tensor. The convolution operation is a linear operation since we simply are multiplying the entries of the input tensor with the weights of the filter and possibly adding a bias if needed. In order to add nonlinearities to the network it is necessary to introduce an activation layer. The activation layer takes as input the output of the convolutional layer and applies a nonlinear transformation to it. The most common activation functions are the the rectified linear (also called Relu), the sigmoid and the hyperbolic tangent function.

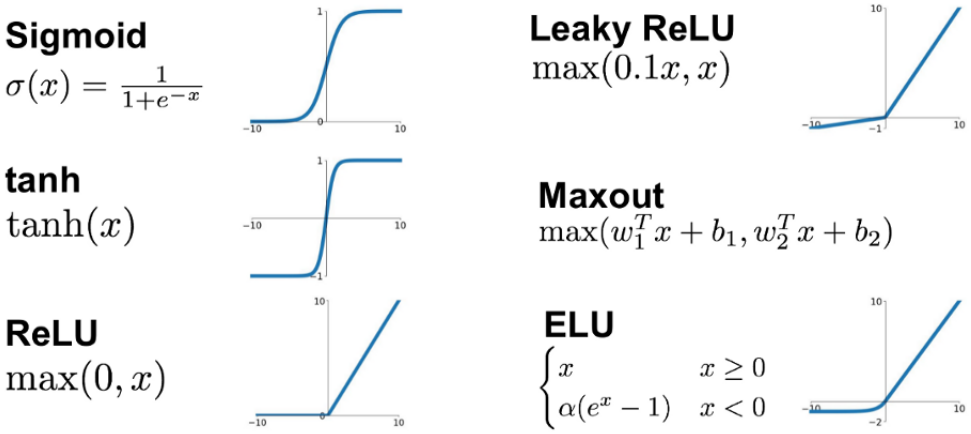


Figure 6.3: Common activation functions (reprinted with permission of [24]).

After having processed the input tensor with a series of convolutional layer, a so-called "Flatten" layer is needed to reshape the output tensor into a vector that can be processed by a fully-connected layer (that is the classical layer of a feed-forward network architecture). In this way it is possible to introduce the final output layer, which is made up as many neurons as the number of classes of the classification problem. In our case, since we are dealing with a binary classification problem, we just need a single

neuron with a sigmoid activation function, so that the output of the network is a numerical value comprised between 0 and 1. At this point a treshold has to be set, in order to transform the output of the network into binary values.

The training process of the network consists in passing the entire training dataset to the network. The training dataset is divided into batches, which are small portion of the dataset on which the network is trained. Therefore the training dataset is not passed to the network all at once, but rather it is passed in smaller quantities called batches. The passage of the entire training dataset to the network is called epoch. The size of the batches and the number of epochs are hyperparameters of the network.
At each epoch, the weights of the filters in the convolution layer and the weights of the fully-connected layers are updated according to the backpropagation algorithm. In this way the network "learns" the correct weights in order to perform a good classification of the samples. The weights updating process takes place according to a gradient descent. The objective is to minimize a given loss function. There are many types of loss function, in our case a suitable choice is the Binary Cross-Entropy loss, defined as follows:

$$H = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i)) \tag{6.1}$$

Where N is the number of samples, y_i is the ground truth of a single sample (true label), and $p(y_i)$ is the predicted probability of a single sample. During the training process, the value of the loss function decreases. The speed with which the gradient descent occurs is given by the learning rate. The learning rate must not be too small, otherwise there would be the risk of getting stuck into a local minimum, but it must not be too big, otherwise the global minimum would never be reached.

A possible network architecture for our classification problmem is depicted in fig. 6.4.

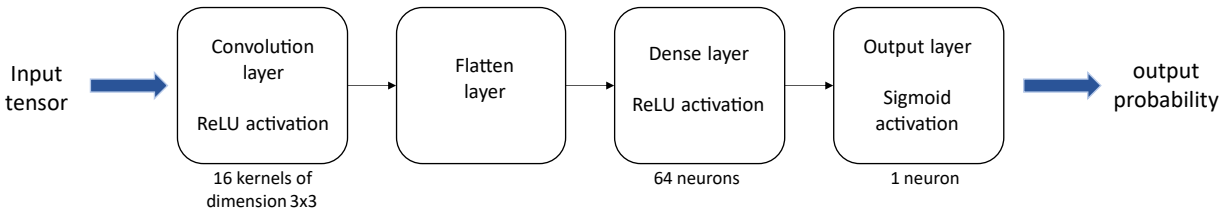


Figure 6.4: Proposed CNN architecture.

The learning rate has been set to 0.001, while the batch size is of 150 samples and the number of epochs are 25. Even in this case the problem of umbalanced data classes is addressed by means of class weights. The class weight in particular is set for the minority class (the class of 0s), and its role is to give more importance to an incorrect classification of the 0s in the definition of the loss function.
For each astrobot in the focal plane, the network is trained using the training dataset of the specific astrobot under analysis. Therefore the approach is similar to the one of the SVM: for each astrobot we

trained the network and then the performances are evaluated using the test dataset of the astrobot. Using a test dataset that is the 10% of the entire dataset in the case of parity-based prediction, and evalutaing the network on a focal plane with 487 astrobots, the results shown in table 6.1 are obtained:

Weight of the 0s	TPR	TNR	Balanced accuracy
3.9	75.62 %	71.92 %	73.77 %
3	80.71 %	65.15 %	72.93 %

Table 6.1: Results on the entire focal plane with the CNN architecture of fig. 6.4 in case of parity-based prediction.

It is possible to observe that the results are very promising in terms of performances on the entire focal plane. However, the use of artificial neural network requires a precise tuning of the parameters involved, including the number of layers, the number of neurons per layer, the number and the dimensions of the kernels and the types of activation functions. A proper setting of these quantities requires a certain level of experience in the field of deep learning. The study of the best architecture suitable for this classification problem requires an in-depth analysis. It was therefore decided to give only one possible idea and introduction on the approach to the problem with the use of a neural network, however demonstrating the potentiality that such a tool offers.

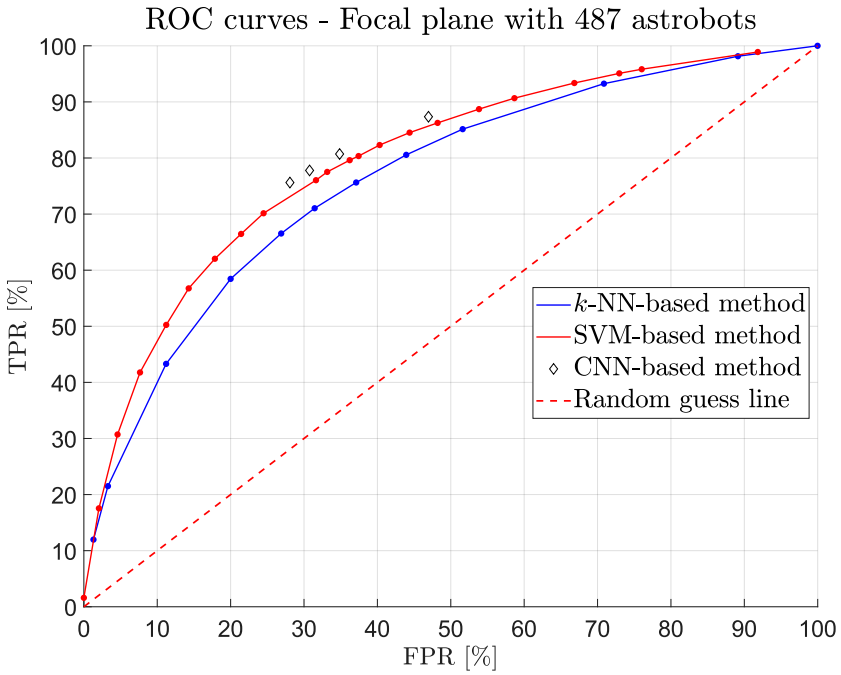


Figure 6.5: ROC curves comparison in the case of parity-based prediction. It is possible to observe that some predictions made using the neural network approach are slightly better than the SVM-based method.

Chapter 7

Conclusion

The main objective of this work was to develop formal methods and algorithms for the prediction of the coordination of each astrobot present in a focal plane. The analysis was carried out by scaling the problem on local units of the focal plane. This approach has proven to be particularly reliable for both the algorithms developed, moreover it has been observed that the performances do not depend on the focal plane extension. Both algorithms have shown good results in terms of a correct classification of the positives and of balanced accuracy. The possibility to embed the categorical information of the parity allowed to relax the assumption of a fixed direction of motion for the second arm of each astrobot in the prediction. Possible future developments could include also other information inside the data, like the safety distance between targets and the vertical arrangement of each astrobot in a focal plane.
The use of a neural network is promising, but requires a fine expertise in the correct chose of the type of network architecture and of the number of parameters which characterize the training process.
Since the most critical condition for prediction is the case of a full neighborhood configuration, it is very important to improve the prediction performances for this specific configuration. For example, a further analysis may focus on which astrobots cause the presence of deadlocks in a given neighborhood, so as to have a further information that can be exploited by the predictor. Since the work shown represents the state of the art on astrobot's convergence prediction, the chances of improving performance can be important.

Acknowledgments

I would first like to thank my supervisor Matin Macktoobian for helping me with useful tips during the development of the algorithms and with the writing of the thesis. I also thank my supervisor in Turin, Marcello Chiaberge, who has always been available to my request and has shown attention to my work. I thank Vittorio Mazzia and Francesco Salvetti for giving me very important advice in developing the neural network.
These five years would not have been so beautiful and full of experiences without the essential support of my family who encouraged me in my choices and helped me financially. I am grateful to my friends, both those from the gang ("la Banda"), and those known in Turin and Lausanne. I consider myself extremely lucky to have entered an environment like that of Collegio Einaudi, which has allowed me to make many friends and to better enjoy my university experience. I don't want to thank all the people I met individually because I'm pretty sure I'd forget someone. However, each person has contributed in some way to my growth and I am grateful for that. I also thank the people I have not met, but who still inspired me with their ideas and stories.
In my opinion, gratitude is one of the best things in this world. I consider it the noblest form of respect for a person. For this reason, I would like to end these acknowledgments with a sentence by the singer Mary Davis: "The more grateful I am, the more beauty I see".

Bibliography

[1] Matin Macktoobian, Denis Gillet, and Jean-Paul Kneib. Complete coordination of robotic fiber positioners for massive spectroscopic surveys. *Journal of Astronomical Telescopes, Instruments, and Systems*, 5(4):045002, 2019.

[2] Dominique Tao, Laleh Makarem, Mohamed Bouri, Jean-Paul Kneib, and Denis Gillet. Priority coordination of fiber positioners in multi-objects spectrographs . In Christopher J. Evans, Luc Simard, and Hideki Takami, editors, *Ground-based and Airborne Instrumentation for Astronomy VII*, volume 10702, pages 2639 – 2652. International Society for Optics and Photonics, SPIE, 2018.

[3] Makarem, Laleh, Kneib, Jean-Paul, Gillet, Denis, Bleuler, Hannes, Bouri, Mohamed, Jenni, Laurent, Prada, Francisco, and Sanchez, Justo. Collision avoidance in next-generation fiber positioner robotic systems for large survey spectrographs. *A&A*, 566:A84, 2014.

[4] Wei Liu and Sanjay Chawla. Class confidence weighted knn algorithms for imbalanced data sets. In Joshua Zhexue Huang, Longbing Cao, and Jaideep Srivastava, editors, *Advances in Knowledge Discovery and Data Mining*, pages 345–356, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[5] Khatijahhusna Abd Rani, Hezlin Aryani Abd Rahman, Simon Fong, Khairudin Zuraida, and Nik Abdullah. An application of oversampling, undersampling, bagging and boosting in handling imbalanced dataset. volume 285, 12 2013.

[6] E. Million. The hadamard product elizabeth million april 12 , 2007 1 introduction and basic results. 2007.

[7] Qingsong Xu, Yi-Zeng Liang, and Yi-Ping Du. Monte carlo cross-validation for selecting a model and estimating the prediction error in multivariate calibration. *Journal of Chemometrics*, 18:112 – 120, 02 2004.

[8] Jiawei Han and Micheline Kamber. *Data mining : concepts and techniques*. Kaufmann, San Francisco [u.a.], 2005.

[9] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[11] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, April 1998.

[12] Bernhard Schölkopf. The kernel trick for distances. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, page 283–289, Cambridge, MA, USA, 2000. MIT Press.

[13] Husam Al-Behadili, Arne Grumpe, Christian Dopp, and Christian Wohler. Non-linear distance based large scale data classifications. pages 613–617, 12 2015.

[14] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013.

[15] Qiubing Ren, Mingchao Li, and Shuai Han. Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives. *Big Earth Data*, pages 1–18, 02 2019.

[16] John Gower. *Similarity, Dissimilarity, and Distance Measure*. 07 2005.

[17] Danilo Bzdok, Martin Krzywinski, and Naomi Altman. Machine learning: Supervised methods. *Nature Methods*, 15, 01 2018.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] Amal Nair. Beginners guide to understanding ROC curve. `https://analyticsindiamag.com/beginners-guide-to-understanding-roc-curve-how-to-find-the-perfect-probability-threshold/`, 2019.

[20] Tom Fawcett. Introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874, 06 2006.

[21] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21, 12 2020.

[22] Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989.

[23] Chaim Baskin, Natan Liss, Avi Mendelson, and Evgenii Zheltonozhskii. Streaming architecture for large-scale quantized neural networks on an fpga-based dataflow platform. 07 2017.

[24] Pawal Jain. A practical guide related to benefits, problems, and comparison of activation functions like Sigmoid, tanh, ReLU, Leaky ReLU and Maxout. `https://mc.ai/complete-guide-of-activation-functions/`, 2019.