

Intelligent distributed systems

Daniele Fontanelli

Department of Industrial Engineering
University of Trento

E-mail address: *daniele.fontanelli@unitn.it*

2022/2023



UNIVERSITÀ
DI TRENTO

Dipartimento di
Ingegneria Industriale

Outline

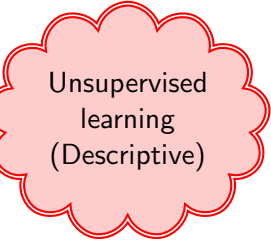
- 1 Machine Learning
- 2 K Nearest Neighbour
- 3 Distributed coverage control
 - Application to actual agents
 - Extensions

Outline


- 1 Machine Learning
- 2 K Nearest Neighbour
- 3 Distributed coverage control
 - Application to actual agents
 - Extensions

Machine Learning


Machine learning tools provide an effective way to detect *patterns* in data.

A red cloud-shaped bubble with a double red outline, containing the text 'Unsupervised learning (Descriptive)'.

Unsupervised
learning
(Descriptive)

An orange cloud-shaped bubble with a double orange outline, containing the text 'Supervised learning (Predictive)'.

Supervised
learning
(Predictive)

A blue cloud-shaped bubble with a double blue outline, containing the text 'Reinforcement Learning'.

Reinforcement
Learning

Machine Learning

Reinforcement learning

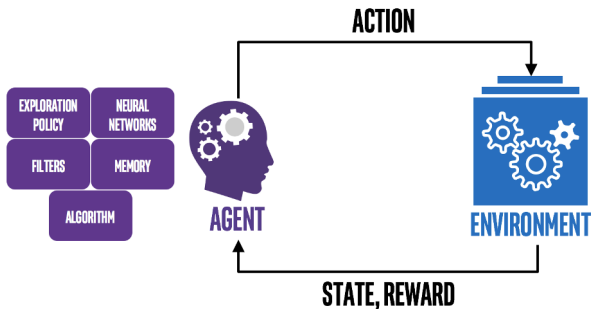


Figure: *Reinforcement Learning* is a subfield of machine learning that teaches an agent how to choose an action from its action space, within a particular environment, in order to maximise rewards over time: *understanding correct behaviour* (courtesy of Dan Lee, “AI³ — Theory, Practice, Business”).

Machine Learning

Unsupervised learning

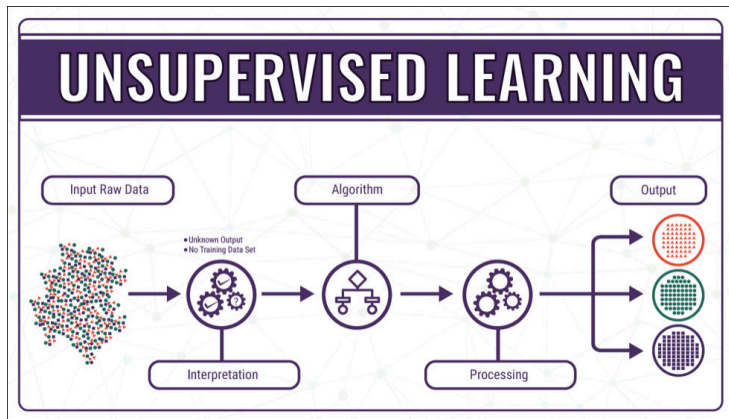


Figure: *Unsupervised Learning* finds interesting patterns in the inputs: *knowledge discovery* (courtesy of Ana Mezic, “Why Unsupervised Machine Learning is the Future of Cybersecurity”, TechNative).

Machine Learning

Supervised learning

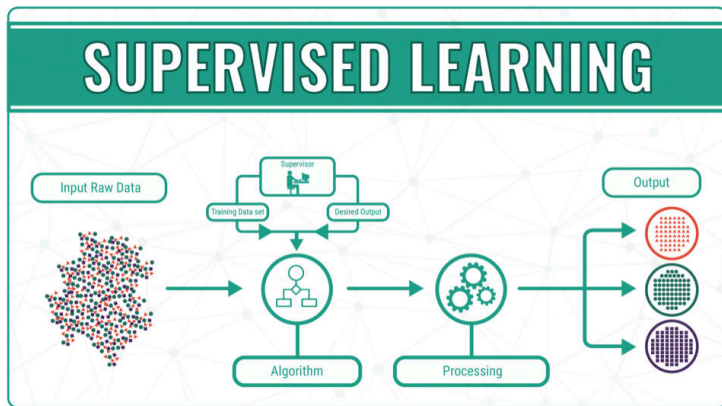


Figure: *Supervised Learning* learns a map from labelled inputs to outputs: *data association* (courtesy of Ana Mezic, “Why Unsupervised Machine Learning is the Future of Cybersecurity”, TechNative).

Machine Learning

Supervised learning

In *supervised learning* we have:

- A set of *inputs* x :
 - Usually it is a set of numbers in an n -dimensional space, i.e. $x \in \mathbb{R}^n$;
 - It can be whatever, such as an image, a written text, etc.;
 - The *dimension* of x depends on the number of *features* (or *attributes* or *covariates*) used to describe it.
- A set of *outputs* y :
 - Usually it is a *number* or a *set of numbers* in an m -dimensional space, i.e. $y \in \mathbb{R}^m$. In this case it is called a *regression*;
 - It is very common also the case in which it is a *categorical variable* in a finite set, i.e. $y \in \{1, \dots, m\}$. In this case it is called *classification* or *pattern recognition*;
 - If it is an *ordered set* (e.g. quality of a product), it is also called a *ordinal regression*.

Machine Learning

Supervised learning vs unsupervised learning

In *supervised learning*, hence, the *training set* is a set of *labelled* examples, i.e. $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^N$.

In *unsupervised learning* we have instead just $\mathcal{S} = \{x_i\}_{i=1}^N$ and the goal is to identify *patterns*.

Machine Learning

Supervised learning: Classification

In this case, the *output* set is $y \in \{1, \dots, m\}$.

- If $m = 2$, then this is called a *binary classification*;
- If $m > 2$, this is a *multi-class classification*;
- The problem of classification can be seen as a *function approximation* problem:
 - The assumption is that the relation is given in the form of $y_i = f(x_i)$;
 - The objective is to derive an *estimate* $\hat{f}(\cdot)$ of the function $f(\cdot)$ using \mathcal{S} ;
 - The estimated function $\hat{f}(\cdot)$ can then be used to make *predictions* on inputs $x \notin \mathcal{S}$. This is also called a *generalisation*.

Machine Learning

Supervised learning: Probabilistic classification

When there is *ambiguity* (i.e. it is not clear if $\hat{f}(x)$ should be $y = i$ or $y = j$) it is useful to resort to *probabilities* and/or *probability distributions*.

- More formally: $p(y|x, \mathcal{S})$ is a *probability density function* if x is continuous or a *probability mass function* if x is discrete;
- Hence, we can derive the *probabilities* $p(a < y \leq b|x, \mathcal{S})$ or $p(y = i|x, \mathcal{S})$;
- Knowing the distributions, we can compute a *best guess* using, for instance, the *Maximum A Posteriori* (MAP), i.e.

$$\hat{y} = \hat{f}(x) = \arg \max_{i \in 1, \dots, m} p(y = i|x, \mathcal{S}).$$

Machine Learning

Supervised learning: Regression

A typical *regression* problem involves $x \in \mathbb{R}$ and $y \in \mathbb{R}$ and happens quite often in engineering problems.

Hence, a *regression* can be seen as a *function approximation* problem as well as *classification*, aka as *spline fit*.

We have seen an example with the *sensor calibration*.

Remember that we can have an estimate of the *probability* of the derived estimates.

Machine Learning

Supervised learning vs unsupervised learning

Let us focus on *supervised learning*.

- We have seen that for the *supervised learning* we have a *training set* $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^N$ that yields a *function approximation* problem with a certain *probability*;
- This is a *conditional density estimation* problem, i.e. $p(y|x, \mathcal{S})$. The approximating function $\hat{f}(\cdot)$ can be given by the MAP estimator;
- In other words, learning $\hat{f}(\cdot)$ is equivalent to learn $p(y_i|x_i, \theta)$, where $(x_i, y_i) \in \mathcal{S}$ and θ are the model parameters.

Machine Learning

Supervised learning vs unsupervised learning: Linear regression

Suppose we have given \mathcal{S} as samples from a line

$$y_i = f(x_i) = a_0 + a_1 x_i, \quad \forall i = 1, \dots, n.$$

It is evident that $n = 2$ it is sufficient to derive the *model parameters* $\theta = (a_0, a_1) = a$, where $a = [a_0, a_1]^T$ (we assume here a *grey box model*). In this case, the estimate $\hat{f}(\cdot) = f(\cdot)$ is *deterministic*.

Notice that we can rewrite the previous relation using a matrix form

$$y_i = f(x_i) = a^T \mathbf{x}_i, \quad \forall i = 1, \dots, n$$

where $\mathbf{x}_i = [1, x_i]^T$.

Machine Learning

Supervised learning vs unsupervised learning: Linear regression

Let us make the model more realistic and let us as suppose that \mathcal{S} is affected by a Gaussian noise (i.e. a **rv**) $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, i.e.

$$y_i = f(x_i) + \varepsilon = a^T \mathbf{x}_i + \varepsilon, \quad \forall i = 1, \dots, n.$$

It turns then out that $y_i \sim \mathcal{N}(a^T \mathbf{x}, \sigma^2)$ (*linear regression* or *inductive bias*).

Therefore, $p(y|x, \theta)$ is a *conditional density* having $\theta = (a, \sigma^2)$.

In this case, the MAP estimator will be given by $y = \hat{f}(x) = a^T \mathbf{x}$.

Machine Learning

Supervised learning vs unsupervised learning: Polynomial regression

This problem can be arbitrarily extended to a generic nonlinear function $\mathbf{g}(x)$. In this case, we have

$$y_i = f(x_i) + \varepsilon = a^T \mathbf{g}(x_i) + \varepsilon, \quad \forall i = 1, \dots, n,$$

where $a = [a_0, a_1, \dots, a_d]^T$ and $\mathbf{g}(x_i) = [1, x_i, x_i^2, \dots, x_i^d]^T$.

In this case, $y_i \sim \mathcal{N}(a^T \mathbf{g}(x_i), \sigma^2)$.

This is also called a *polynomial regression*.

Machine Learning

Useful concepts

- If the model to estimate has a *fixed number of parameters*, it is called a *parametric model* (e.g. *linear regression*), *nonparametric* otherwise (e.g. *K nearest neighbour*).
- An important concept in learning is the *overfitting*: this happens when we try to model all the possible variations in the input, which is wrong because most likely we are modelling the noise! (This phenomenon can be easily proved with *linear regression*).

Outline

- 1 Machine Learning
- 2 K Nearest Neighbour
- 3 Distributed coverage control
 - Application to actual agents
 - Extensions

K Nearest Neighbour

K-nearest neighbours is probably the simplest *non-parametric classifier* that can be defined.

KNN searches for the k points that are closer to a test point q , count how many members of a certain class belong to the set and compute the related probability using a *frequentist* approach.

K Nearest Neighbour

Let $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^N$ the *training data* that is *labelled* and $y \in \{1, \dots, m\}$ the classes.

Let $\alpha(q, \mathcal{S}, k)$ be set of k *point indices* of \mathcal{S} that are closer to the test point q .

Let \mathbb{I} be the *indicator function*

$$\mathbb{I}(a) = \begin{cases} 1, & \text{if } a \text{ is true,} \\ 0, & \text{if } a \text{ is false.} \end{cases}$$

We finally have a *multivariate pmf*:

$$p(y = i | q, \mathcal{S}, k) = \sum_{i \in \alpha(q, \mathcal{S}, k)} \frac{\mathbb{I}(y = i)}{k}.$$

K Nearest Neighbour

Usually, the distance to compute the KNN is the *Euclidean distance*, but the method can be applied to *any distance definition*.

K Nearest Neighbour

KNN: example

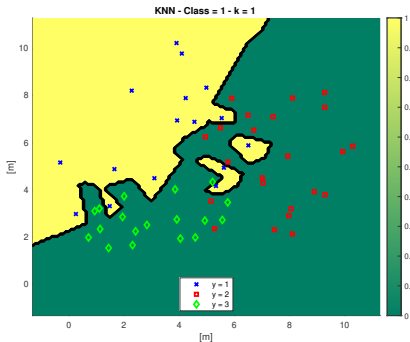
Let us make an example. Let us suppose to have $m = 3$ labels for points on a plane, i.e. $x_i \in \mathbb{R}^2$.

Let us suppose to have $\mathcal{S}_1 = \{x_i, 1\}$, $i = 1, \dots, 16$, $\mathcal{S}_2 = \{x_i, 2\}$, $i = 1, \dots, 28$ and $\mathcal{S}_3 = \{x_i, 3\}$, $i = 1, \dots, 18$. Hence $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$.

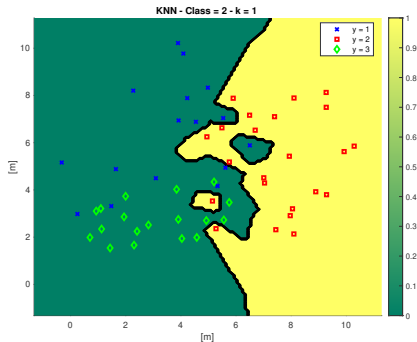
Moreover, $x_i \in \mathcal{S}_1$ are $x_i \sim \mathcal{U}([-1, 1], [7, 11])$; $x_i \in \mathcal{S}_2$ are $x_i \sim \mathcal{U}([5, 2], [11, 9])$ and $x_i \in \mathcal{S}_3$ are $x_i \sim \mathcal{U}([0, 1], [6, 5])$.

K Nearest Neighbour

KNN: example



(a)

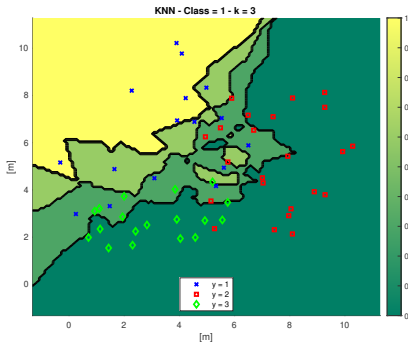


(b)

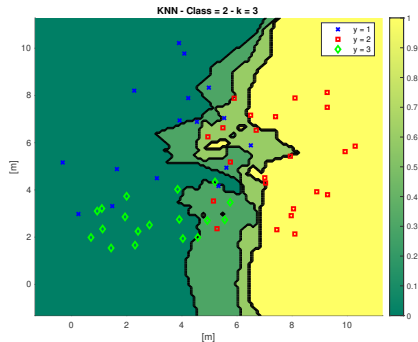
Figure: Conditional probability $p(y = i|q, \mathcal{S}, k)$ with $k = 1$ for $y = 1$ (a) and $y = 2$ (b). Of course, the value for $y = 3$ can be easily derived since the pmf sums up to one.

K Nearest Neighbour

KNN: example



(a)

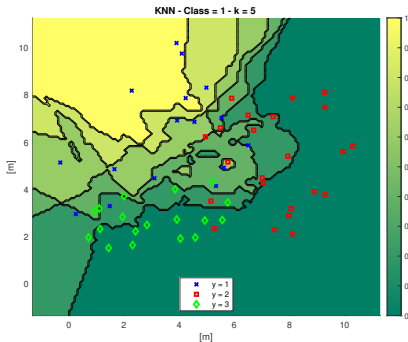


(b)

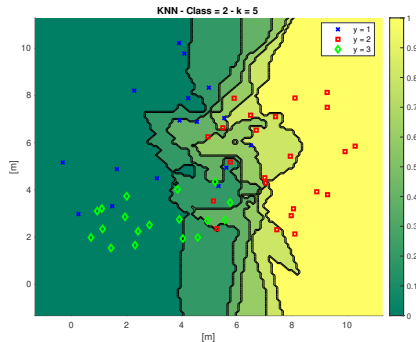
Figure: Conditional probability $p(y = i|q, \mathcal{S}, k)$ with $k = 3$ for $y = 1$ (a) and $y = 2$ (b). Of course, the value for $y = 3$ can be easily derived since the pmf sums up to one.

K Nearest Neighbour

KNN: example



(a)

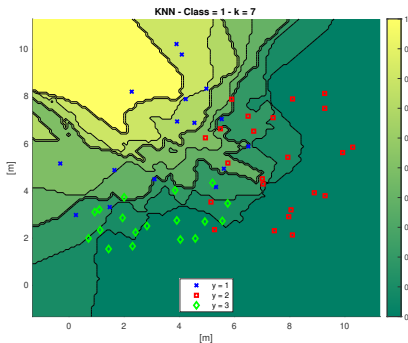


(b)

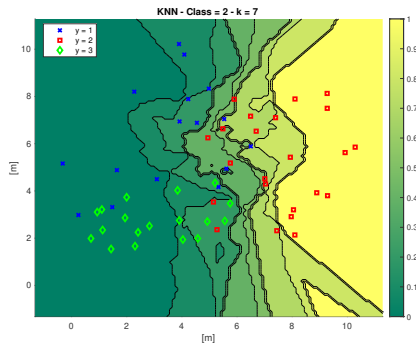
Figure: Conditional probability $p(y = i|q, \mathcal{S}, k)$ with $k = 5$ for $y = 1$ (a) and $y = 2$ (b). Of course, the value for $y = 3$ can be easily derived since the pmf sums up to one.

K Nearest Neighbour

KNN: example



(a)

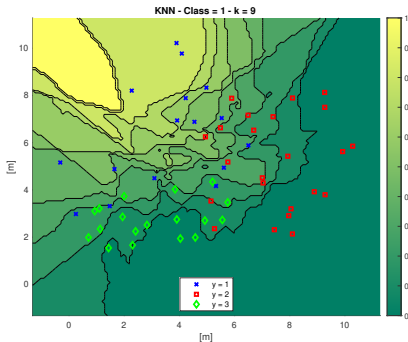


(b)

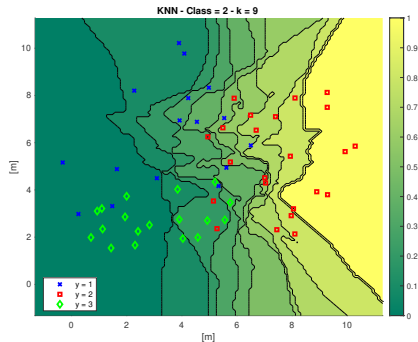
Figure: Conditional probability $p(y = i|q, \mathcal{S}, k)$ with $k = 7$ for $y = 1$ (a) and $y = 2$ (b). Of course, the value for $y = 3$ can be easily derived since the pmf sums up to one.

K Nearest Neighbour

KNN: example



(a)

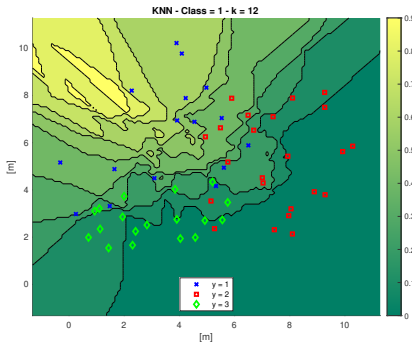


(b)

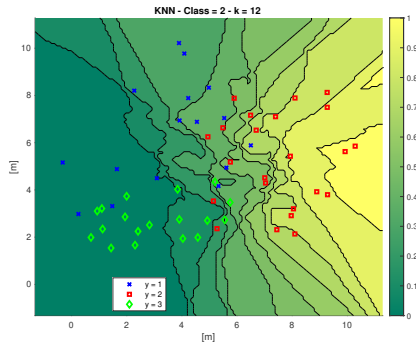
Figure: Conditional probability $p(y = i|q, \mathcal{S}, k)$ with $k = 9$ for $y = 1$ (a) and $y = 2$ (b). Of course, the value for $y = 3$ can be easily derived since the pmf sums up to one.

K Nearest Neighbour

KNN: example



(a)

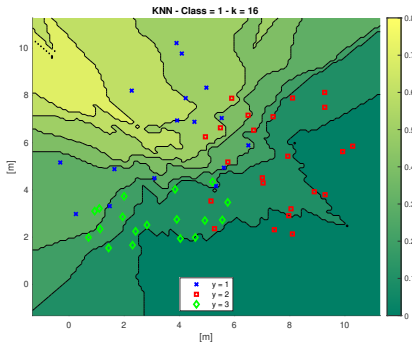


(b)

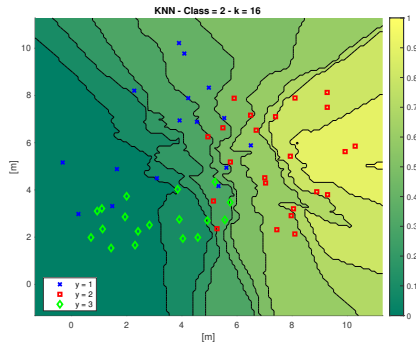
Figure: Conditional probability $p(y = i|q, \mathcal{S}, k)$ with $k = 12$ for $y = 1$ (a) and $y = 2$ (b). Of course, the value for $y = 3$ can be easily derived since the pmf sums up to one.

K Nearest Neighbour

KNN: example



(a)



(b)

Figure: Conditional probability $p(y = i|q, \mathcal{S}, k)$ with $k = 16$ for $y = 1$ (a) and $y = 2$ (b). Of course, the value for $y = 3$ can be easily derived since the pmf sums up to one.

K Nearest Neighbour

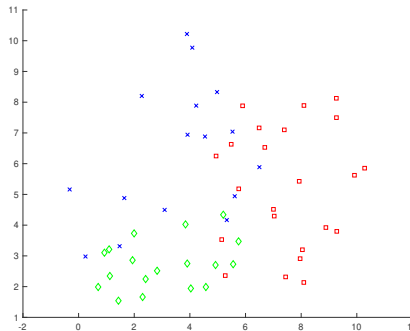
Remark

A KNN classifier with $k = 1$ induces a **Voronoi tessellation**: at any point $x_i \in \mathcal{S}$ (with $x_i \in \mathbb{R}^l$) is associated a region $V(x_i)$ such that:

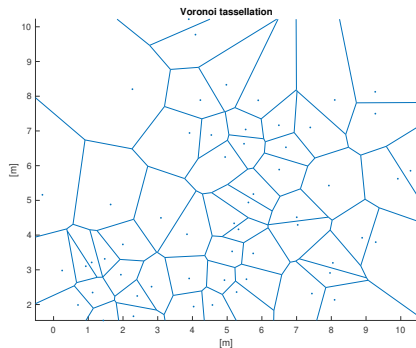
$$V(x_i) = \{q \in \mathbb{R}^l \mid \|q - x_i\| \leq \|q - x_j\|, \forall x_j \neq x_i \text{ and } x_j \in \mathcal{S}\}.$$

K Nearest Neighbour

KNN: example



(a)



(b)

Figure: Given the *training data* (a) we can easily obtain the Voronoi tessellation (b) for the same example described previously.

K Nearest Neighbour

Given the previous probabilities retrieved from the KNN, we can now apply the *Maximum A Posteriori* (MAP) to obtain the *best guess*:

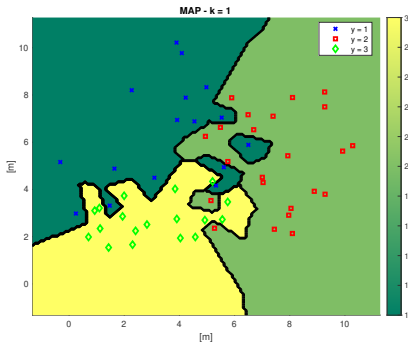
$$\hat{y} = \hat{f}(x) = \arg \max_{i \in 1, \dots, m} p(y = i | x, \mathcal{S}, k),$$

which is a function of the chosen k .

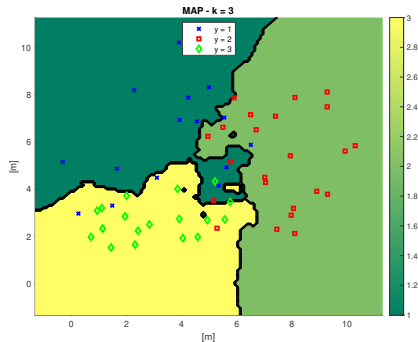
Here comes some results on the previous example.

K Nearest Neighbour

KNN: example



(a)

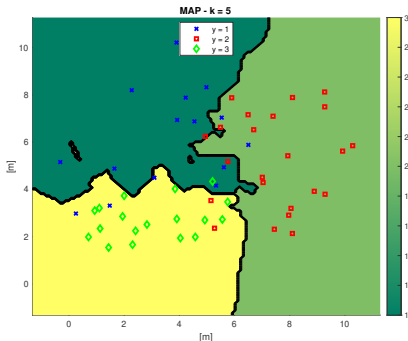


(b)

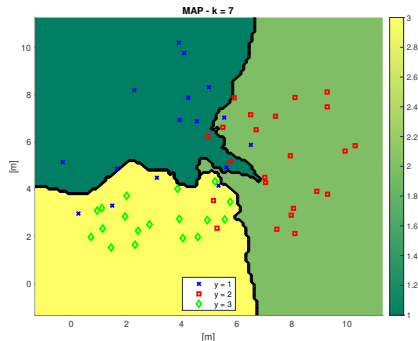
Figure: MAP estimator with $k = 1$ (a) and $k = 3$ (b).

K Nearest Neighbour

KNN: example



(a)

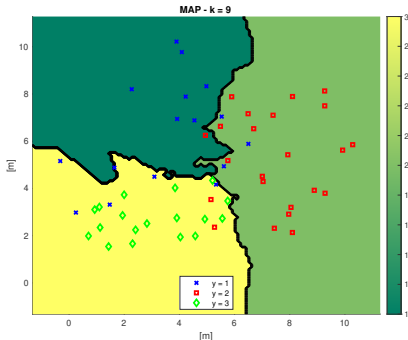


(b)

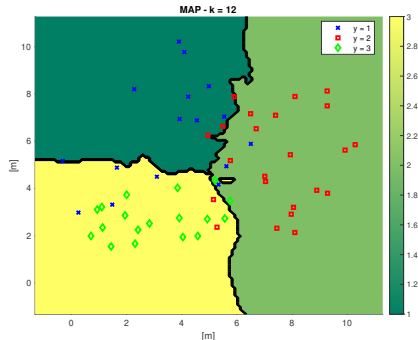
Figure: MAP estimator with $k = 5$ (a) and $k = 7$ (b).

K Nearest Neighbour

KNN: example



(a)

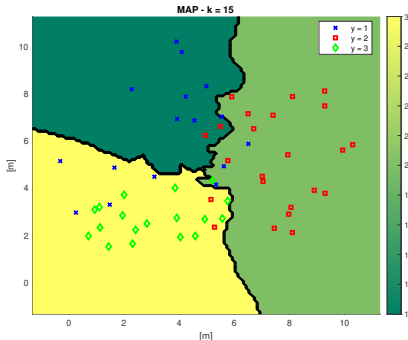


(b)

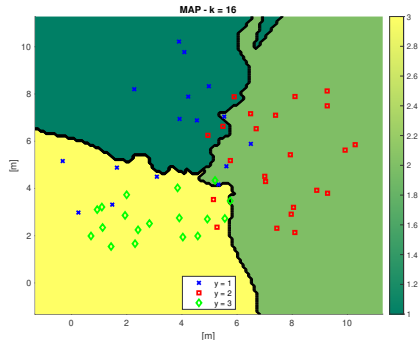
Figure: MAP estimator with $k = 9$ (a) and $k = 12$ (b).

K Nearest Neighbour

KNN: example



(a)



(b)

Figure: MAP estimator with $k = 15$ (a) and $k = 16$ (b).

Outline

- 1 Machine Learning
- 2 K Nearest Neighbour
- 3 Distributed coverage control
 - Application to actual agents
 - Extensions

K Nearest Neighbour

Voronoi tessellation

Consider a mission space $\mathcal{Q} \in \mathbb{R}^2$ and a set of n *agents* (i.e., robots, nodes, sensors), we identify the i -th agent position with $p_i = [x_i, y_i]^T$. The i -th Voronoi cell is defined as

$$\mathcal{V}_i = \{q \in \mathcal{Q} \mid \|q - p_i\| \leq \|q - p_j\|, \forall j \neq i\},$$

where $q \in \mathcal{Q}$ is a generic point belonging to the mission space. In other words, the distance between the i -th agent position p_i and each point q that belongs to the i -th Voronoi cell is smaller or equal than the distance between q and any other node position p_j . We consider the following first order dynamics for each agent

$$\dot{p}_i = u_i, \quad \forall i = 1, \dots, n.$$

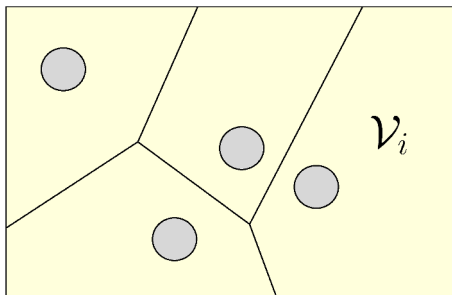


Figure: Voronoi tessellation example.

K Nearest Neighbour

Voronoi tessellation: centroid

We consider the expression for the *mass* of the i -th Voronoi cell

$$M_{\mathcal{V}_i} = \int_{\mathcal{V}_i} dq.$$

Moreover, we consider the expression of the *centroid position* of the i -th Voronoi cell

$$C_{\mathcal{V}_i} = \frac{1}{M_{\mathcal{V}_i}} \int_{\mathcal{V}_i} q dq.$$

K Nearest Neighbour

Voronoi tessellation: Lloyd control

It is then defined to define the following *coverage cost metric*

$$J_{\text{cov}}(p, \mathcal{V}) = \sum_{i=1}^n \int_{\mathcal{V}_i} \|q - p_i\|^2 dq,$$

which relates to a *static coverage problem*: *the system reaches a stable configuration that deploys the agents in the mission space so that a coverage metric (e.g. $J_{\text{cov}}(p, \mathcal{V})$) is optimised.*

Of course, the minimum is $J_{\text{cov}}(p, \mathcal{V})$, the higher is the coverage.

K Nearest Neighbour

Voronoi tessellation: Lloyd control

The following Lloyd-based proportional control law

$$\dot{p}_i(\mathcal{V}_i) = -k_p (p_i - C_{\mathcal{V}_i}),$$

where $k_p > 0$ is a tuning parameter, ensures the *convergence* of each agent on its *Voronoi partition centroid*, thus ensuring the *optimal deployment for the coverage problem*.

The proof is technically given by using $J_{\text{cov}}(p, \mathcal{V})$ as a *Lyapunov function* and then by applying the LaSalle's invariance principle (see J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," IEEE Transactions on robotics and Automation, vol. 20, no. 2, pp. 243–255, 2004 for details).

Outline

- 1 Machine Learning
- 2 K Nearest Neighbour
- 3 Distributed coverage control
 - Application to actual agents
 - Extensions

K Nearest Neighbour

Voronoi tessellation

Let us consider the n nodes as a *mobile sensor network*, with limited sensing range R_s and communication range R_c for each sensor, and let us assume that $R_c > 2R_s$.

In this case, we can define the i -th *sensing range limited Voronoi cell* as the intersection of the Voronoi cell \mathcal{V}_i and the *closure* of the sensing range circle, which we indicate with $\bar{\mathcal{C}}_{i,R_s}$, i.e.

$$\bar{\mathcal{V}}_i = \{ \mathcal{V}_i \cap \bar{\mathcal{C}}_{i,R_s} \}.$$

K Nearest Neighbour

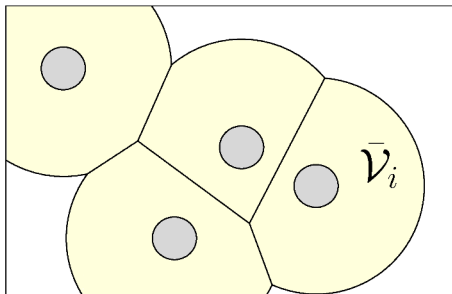


Figure: The Sensing range limited Voronoi tessellation.

K Nearest Neighbour

Voronoi tessellation

The construction of the sensing range limited Voronoi diagram can be easily done in a *distributed fashion* under the condition $R_c > 2R_s$, i.e. each agent computes its own Voronoi cell \mathcal{V}_i only *on the basis of the neighbours positions*.

The neighbours set is defined as

$$\mathcal{N}_i = \{p_j \in \mathbb{R}^2 \mid \|p_i - p_j\| < R_c\}.$$

Indeed, when the i -th agent knows the neighbours positions, it can test all the points $q \in \bar{\mathcal{C}}_{i,R_s}$, if the condition $\|q - p_i\| \leq \|q - p_j\|, \forall j \in \mathcal{N}_i$ is verified, then the point q belongs to the i -th cell.

Outline

- 1 Machine Learning
- 2 K Nearest Neighbour
- 3 Distributed coverage control
 - Application to actual agents
 - Extensions

Voronoi based distributed control

Static obstacles

The previous control law works fine in a *convex space*, i.e. without the presence of obstacles.

Let \mathcal{O} be the *obstacle set* and let \mathcal{S}_i be the i -th agent *visibility set*, i.e.

$$\mathcal{S}_i = \{q \in \mathcal{Q} \mid p_i - \gamma(p_i - q) \notin \mathcal{O}\}, \quad \forall \gamma \in [0, 1],$$

line of sight space.

By intersect the visibility set with $\bar{\mathcal{V}}_i$, we obtain the *Voronoi-visible set*

$$\mathcal{W}_i = \{\bar{\mathcal{V}}_i \cap \mathcal{S}_i\}.$$

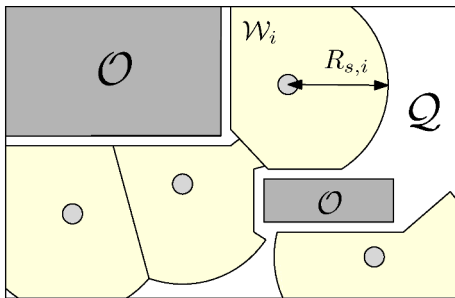


Figure: Voronoi-Visible tessellation.

Notice that the *encumbrance of the agents* is managed by *inflating the obstacle dimensions* of an amount equal to the occupancy radius of the agent.

Voronoi based distributed control

Static obstacles

We can now compute the Lloyd-based control law on this set, i.e.

$$\dot{p}_i(\mathcal{W}_i) = -k_p (p_i - C_{\mathcal{W}_i}).$$

Voronoi based distributed control

Dynamic obstacles

Further modifications ensures *absence of collisions between the agents of the group and with other moving obstacles*.

To this end, assume that the i -th agent can be approximated with a circle of radius δ_i . Hence the Voronoi tessellation can be modified as follows

$$\tilde{\mathcal{V}}_i = \begin{cases} \{q \in \mathcal{Q} \mid \|q - p_i\| \leq \|q - p_j\|\}, & \text{if } \Delta_{ij} \leq \frac{\|p_i - p_j\|}{2} \\ \{q \in \mathcal{Q} \mid \|q - p_i\| \leq \|q - \tilde{p}_j\|\}, & \text{otherwise} \end{cases}$$

where $\Delta_{ij} = \delta_j + \delta_i$ and $\tilde{p}_j = p_j + 2 \left(\Delta_{ij} - \frac{\|p_i - p_j\|}{2} \right) \frac{p_i - p_j}{\|p_i - p_j\|}$.

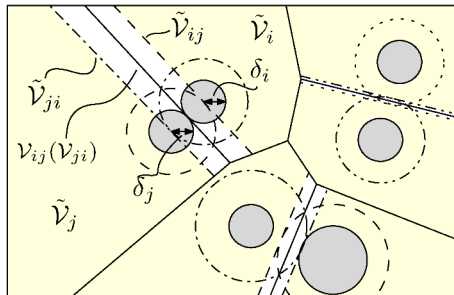


Figure: Modified Voronoi tessellation to account for dynamic obstacles.

Voronoi based distributed control

Desired goal positions

To ensure that the agents move towards a *desired goal position*, we can modify the concept of *distance* adopted in the Voronoi tessellation. Consider *pdf-like weighting factor* $\varphi(q)$ defined in the operational space \mathcal{Q} , i.e it can be a *bivariate Gaussian pdf* with mean value centred in the desired position.

Voronoi based distributed control

Desired goal positions

Hence, we can modify the idea of *mass* and centroid given previously as:

$$M_{\mathcal{V}_i} = \int_{\mathcal{V}_i} \varphi(q) dq,$$

i.e. the *normalisation factor of a pdf* computed in the cell \mathcal{V}_i .

Moreover, we consider the expression of the *centroid position* of the i -th Voronoi cell

$$C_{\mathcal{V}_i} = \frac{1}{M_{\mathcal{V}_i}} \int_{\mathcal{V}_i} \varphi(q) q dq,$$

i.e. the *average value* computed in the cell \mathcal{V}_i .

The extension to static and dynamic obstacles applies then straightforwardly.

Voronoi based distributed control

Example: our Shelfy robots



Figure: Safe navigation in complex environment with three robots in the DII department.

Voronoi based distributed control

Further extension

Given the free mission space $\mathcal{Q} \in \mathbb{R}^2$, the obstacle space $\mathcal{O} \in \mathbb{R}^2$ and a finite number n of robotic agents with initial position $p_i(0) \in \mathcal{Q} \ \forall i = 1 \dots n$, it is possible to design control laws that operate through local interactions so that the collective (global) behaviour of the group *additionally* fulfils one of the following goals:

- *Connectivity maintenance*: given an initial connectivity graph defined by edges and vertices, the robots move so that the number of edges in the connectivity graph is a non-decreasing function of time and if two vertices are linked through an edge, they remain so;
- *Exploration*: the robot motion strategy guarantees that the entire mission space is covered by the sensing range of the robots in finite time; if the number of robots is not sufficient to cover the space, the different locations can be covered at different times;

Voronoi based distributed control

Further extension

- *Navigation*: i) Flocking: move the robotic agents from the starting positions $\mathcal{P} = \{p_1(0), p_2(0), \dots, p_n(0)\}$ to the goal positions $\mathcal{E} = \{e_1, e_2, \dots, e_n\}$; ii) Formation: move the robotic agents from the starting positions $\mathcal{P} = \{p_1(0), p_2(0), \dots, p_n(0)\}$ to the goal positions $\mathcal{E} = \{e_1, e_2, \dots, e_n\}$ keeping constant pairwise distance between the agents;
- *Rendezvous*: each robot of the system converges to the same rendezvous position.