Racetrack programming

(rif.: “Reinforcement Learning: An Introduction” second edition Richard S. Sutton and Andrew G. Barto - The MIT Press - 2020 - http://incompleteideas.net/book/RLbook2020.pdf)

Si considera il problema di guidare una automobile su una pista da corsa, discretizzata su un foglio a quadretti.
L’obiettivo è quello di raggiungere il traguardo nel minor tempo possibile, ovvero con il minor numero di mosse possibili, senza uscire dalla pista.
L’automobile in ogni istante si trova in una posizione della pista e ha una certa velocità e direzione di marcia.
Ad ogni istante temporale, il pilota decide l’azione da compiere per variare velocità e direzione dell’auto.
Le azioni possibili sulla velocità sono accelera, rallenta, continua così le quali, rispettivamente aumentano di 1, diminuiscono di 1, lasciano invariata la velocità dell’auto. La velocità varia da 1 a 4 quadretti per istante temporale.
Le azioni sulla direzione sono gira a dx e gira a sx, che cambiano la direzione di 90° mantenendo inalterata la velocità.
Ogni episodo di corsa inizia in un punto scelto a caso della linea di partenza e termina quando l’auto attraversa la linea del traguardo.
La ricompensa è negativa, -1, per ogni mossa che non giunge al traguardo, -100 per penalità quando si va fuori pista e +100 quando si giunge al traguardo.

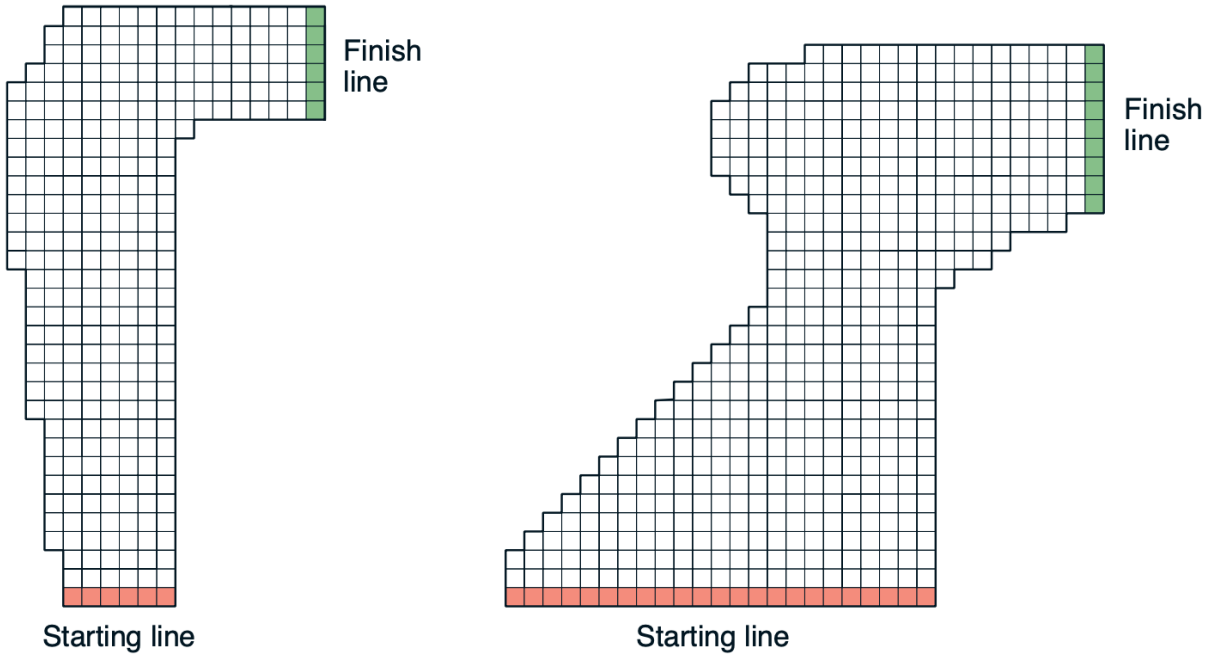
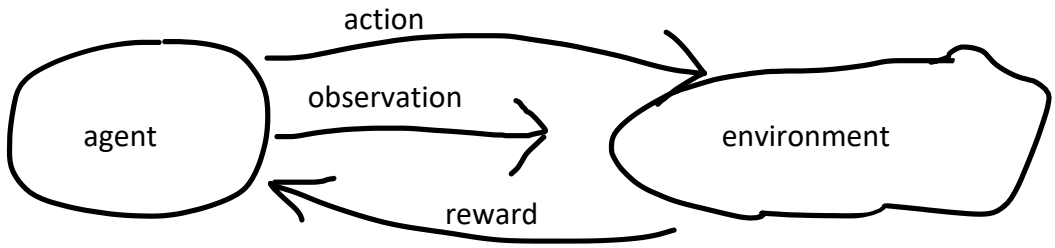


Figura: un paio di esempi di pista

Reinforcement Learning



L'agente agisce sull'ambiente, ne osserva i cambiamenti e riceve delle ricompense per il suo operato.
Le ricompense (reward) possono essere positive o negative (penalità)
L'obiettivo è trovare la policy (politica) che consente di massimizzare il totale delle ricompense ricevute.

La policy è una funzione dello stato in cui si trova l'auto (posizione nella pista e i suoi parametri di moto) che fornisce al pilota l'azione da compiere.

I punti di reward sono: al traguardo +100, quando si esce di pista -100, ad ogni mossa -1.
Grazie a quest'ultima penalità si persegue l'individuazione delle azioni che consentono di arrivare al traguardo nel minor numero di mosse.
Le azioni sono i comandi dell'auto: accelera, rallenta, continua così, gira a dx, gira a sx

Ci sono tecniche di Reinforcement Learning che si basano sul modello dell'ambiente, ovvero conoscono perfettamente come è fatta la pista. Cosicché, dato lo stato, ovvero le coordinate x, y dove si trova l'auto e la sua velocità e direzione di moto, si può determinare l'azione ottimale da compiere.
Queste tecniche non vanno bene per questo problema perché porterebbero ad apprendere il comportamento di guida solo con la pista su cui viene effettuato l'apprendimento.
Esse vanno bene per quelle situazioni, come il gioco degli scacchi, dove la scacchiera è sempre la stessa.

Poi ci sono le tecniche cosiddette "model free", che sono di uso più generale.
Esse richiedono l'effettuazione di "episodi completi", ovvero sequenze di azioni dall'inizio alla fine della corsa (traguardo o fuori pista), e poi, analizzando tutte le mosse effettuate, aggiornano la policy.
In questo modo, dopo un gran numero di episodi, si arriva ad ottenere la "policy ottima".
Queste tecniche vengono utilizzate anche quando, pur conoscendolo, il modello prevede un numero troppo elevato di possibili stati da studiare.

Per rappresentare lo "stato" ci si basa su informazioni locali dell'ambiente dove si trova l'agente: in pratica si considerano il tratto di pista che si vede di fronte all'auto, la velocità e la direzione della stessa.
A tal fine, si prende una immagine dell'ambiente che si trova di fronte all'auto.
Tuttavia, una immagine contiene troppi dettagli e troppi particolari inutili che risultano di disturbo per l'interpretazione della stessa.
Pertanto conviene estrarre le caratteristiche importanti (feature) da quello che si vede, per capire se la strada è diritta oppure c'è una curva: si potrebbero considerare, ad esempio, la presenza di un cartello stradale o la forma della linea bianca di mezzeria o l'andamento dei bordi della pista che si trovano di fronte.
Considerando le feature si ottiene una specie di approssimazione dell'ambiente in cui ci si trova e questo consente di ridurre di molto il numero di possibili stati da considerare.

Per effettuare il processo di apprendimento si usa la tecnica Qlearning, che utilizza una tabella con i valori Q (qualità) basati sul totale delle ricompense che si ottengono se in un certo stato si effettua una determinata azione.

	azioni				
Q	Accelera	Rallenta	Continua	Gira dx	Gira sx
Stato0					
Stato1					
Stato2					
…					

All'inizio la tabella contiene tutti valori nulli.
Poi si aggiornano i valori della stessa con la formula di Bellman

$$Q_{new}(S_t, A_t) = (1 - \alpha)\, Q(S_t, A_t) + \alpha\, (R_{t+1} + \gamma\, Max_a\, Q(S_{t+1}, a))$$

Dove α = fattore di apprendimento (learning rate) = 0.9
γ = fattore di sconto = 1
t è il tempo, ovvero l'istante temporale in cui viene effettuata ciascuna mossa

Pertanto, il nuovo valore di $Q(S_t, A_t)$ viene calcolato come somma di due contributi: in parte il vecchio valore $Q(S_t, A_t)$ e in parte il valore derivante dagli esiti dell'ultimo episodio $R_{t+1} + \gamma\, Max_a\, Q(S_{t+1}, a)$

Per poter applicare la suddetta formula, occorre memorizzare per ogni episodio la sequenza di mosse effettuate: $(S_t, A_t, R_{t+1}\ S_{t+1})$ e la ricompensa ottenuta R_{t+1}
Al termine dell'episodio si procede a ritroso, dall'ultima mossa fino alla prima, e si aggiornano i valori Q della tabella, utilizzando la formula di Bellman.

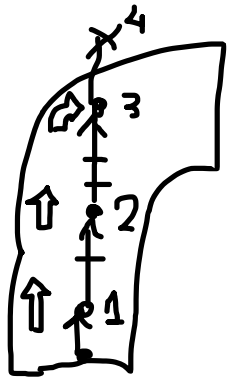
L'apprendimento richiede un gran numero di episodi, affinché tutti gli stati e tutte le azioni siano stati sperimentati un adeguato numero di volte.
Si può calcolare la variazione di valore che subisce ogni cella (in valore assoluto)

$$delta = |Q_{new}(S_t, A_t) - Q(S_t, A_t)|$$

e ci si ferma quando in un episodio il massimo valore di delta è minore di un certo valore di soglia (ad esempio 0.1). Questo indica che i valori della matrice Q si sono stabilizzati e forniscono la policy ottima.

Esempio di aggiornamento della tabella Q

Episodio 1



Stato 1: cartello diritto, velocità 1, direzione su
Azione 1: accelera (la velocità da 1 diventa 2) Reward: -1

Stato 2: cartello diritto, velocità 2, direzione su
Azione 2: accelera (la velocità da 2 diventa 3) Reward = -1

Stato 3: cartello curva, velocità 3, direzione su
Azione 3: rallenta (la velocità da 3 diventa 2) Reward = -100

Stato 4: fine corsa per schianto fuori pista

Tabella iniziale

Q	Accelera	Rallenta	Gira dx	…
Stato1: Cartello diritto Velocità 1, Direzione su	0	0	0	
Stato2: Cartello diritto Velocità 2, Direzione su	0	0	0	
Stato3: Cartello curva Velocità 3, Direzione su	0	0	0	
Stato4: fine episodio	0	0	0	

Applico la formula di Bellman con Stato3, Azione3=rallenta, Reward=-100, Stato4
Q(stato3, rallenta) = (1 - 0.9) 0 + (0,9) (-100 + 0) = -90

Si ottiene

Q	accelera	rallenta	Gira dx	…
Stato1: Cartello diritto Velocità 1, Direzione su	0	0	0	
Stato2: Cartello diritto Velocità 2, Direzione su	0	0	0	
Stato3: Cartello curva Velocità 3, Direzione su	0	**-90**	0	
Stato4: fine episodio	0	0	0	

Applico la formula di Bellman con Stato2, Azione2=accelera, Reward=-1, Stato3
Q(stato2, accelera) = (1 - 0.9) 0 + (0,9) (-1 + 0) = -0.9

Si ottiene

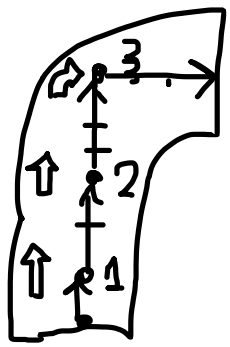
Q	accelera	rallenta	Gira dx	…
Stato1: Cartello diritto Velocità 1, Direzione su	0	0	0	
Stato2: Cartello diritto Velocità 2, Direzione su	**-0.9**	0	0	
Stato3: Cartello curva Velocità 3, Direzione su	0	-90	0	
Stato4: fine episodio	0	0	0	

Applico la formula di Bellman con Stato1, Azione1=accelera, Reward=-1, Stato2
Q(stato1, accelera) = (1 - 0.9) 0 + (0,9) (-1 + 0) = -0.9

Si ottiene

Q	accelera	rallenta	Gira dx	…
Stato1: Cartello diritto Velocità 1, Direzione su	**-0.9**	0	0	
Stato2: Cartello diritto Velocità 2, Direzione su	-0.9	0	0	
Stato3: Cartello curva Velocità 3, Direzione su	0	-90	0	
Stato4: fine episodio	0	0	0	

Episodio 2



Stato 1: cartello diritto, velocità 1, direzione su
Azione 1: accelera (la velocità da 1 diventa 2) Reward: -1

Stato 2: cartello diritto, velocità 2, direzione su
Azione 2: accelera (la velocità da 2 diventa 3) Reward = -1

Stato 3: cartello curva, velocità 3, direzione su
Azione 3: gira dx (la velocità rimane 3) Reward = +100

Stato 4: fine corsa per raggiunto traguardo

Applico la formula di Bellman con Stato3, Azione3=gira dx, Reward=+100, Stato4

Q(stato3, gira dx) = (1 - 0.9) 0 + (0,9) (+100 + 0) = +90

Si ottiene

Q	accelera	rallenta	Gira dx	…
Stato1:Cartello diritto Velocità 1, Direzione su	-0.9	0	0	
Stato2: Cartello diritto Velocità 2, Direzione su	-0.9	0	0	
Stato3: Cartello curva Velocità 3, Direzione su	0	-90	**+90**	
Stato4: fine episodio	0	0	0	

Applico la formula di Bellman con Stato2, Azione2=accelera, Reward=-1, Stato3
Q(stato2, accelera) = (1 - 0.9) (-0.9) + (0,9) (-1 + 90) = +81

Si ottiene

Q	accelera	rallenta	Gira dx	…
Stato1: Cartello diritto Velocità 1, Direzione su	-0.9	0	0	
Stato2: Cartello diritto Velocità 2, Direzione su	**+81**	0	0	
Stato3: Cartello curva Velocità 3, Direzione su	0	-90	+90	
Stato4: fine episodio	0	0	0	

Applico la formula di Bellman con Stato1, Azione1=accelera, Reward=-1, Stato2
Q(stato1, accelera) = (1 - 0.9) (-0.9) + (0,9) (-1 + 81) = +71

Si ottiene

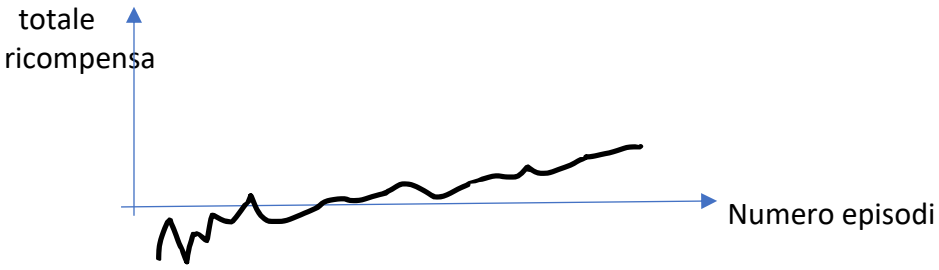
Q	accelera	rallenta	Gira dx	…
Stato1: Cartello diritto Velocità 1, Direzione su	**+71**	0	0	
Stato2: Cartello diritto Velocità 2, Direzione su	+81	0	0	
Stato3: Cartello curva Velocità 3, Direzione su	0	-90	+90	
Stato4: fine episodio	0	0	0	

Scelta dell'azione da compiere

La tabella fornisce per ogni stato i punteggi Q delle diverse possibili azioni.
Tale tabella non contiene subito i valori definitivi di Q ma viene via via aggiornata grazie al suddetto meccanismo di apprendimento.
Alla fine i valori Q convergeranno ai valori che esprimono il corretto valore di ciascuna azione in ciascuno stato.
Con i valori definitivi di Q, la policy ottima consiste nello scegliere per ciascuno stato l'azione che ha il punteggio Q maggiore.
Con i valori provvisori, invece, non conviene limitarsi a questo criterio di "exploitation" (sfruttamento) della conoscenza fin qui acquisita, bensì conviene attuare anche un meccanismo di "exploration" (esplorazione) delle diverse possibili azioni.
Pertanto, durante la fase di apprendimento si procede scegliendo con probabilità ε una azione random (exploration), e con probabilità (1 - ε) l'azione con il punteggio Q più elevato (exploitation); a parità di valore di Q la scelta tra le diverse azioni è arbitraria.
Il valore del parametro ε inizia con un valore elevato, ad esempio $\varepsilon = 0.50$, e viene fatto diminuire man mano che si effettuano episodi di apprendimento. Ad esempio lo si può diminuire di 0.0001 ad ogni episodio: in questo modo si effettueranno 5000 episodi prima di considerare definitiva la tabella.

Diagnostica del processo di apprendimento

Può risultare utile visualizzare un grafico che illustri gli esiti del processo di apprendimento.
Nell'asse delle x si mette il numero di episodi usati per l'apprendimento e nell'asse y il totale delle ricompense (reward) ottenute da ciascun episodio.



Si potrebbe anche monitorare la convergenza dei valori di Q della tabella mettendo nell'asse y il massimo valore di delta delle celle aggiornate durante l'epiosodio; delta = valore assoluto della variazione di valore di una cella.

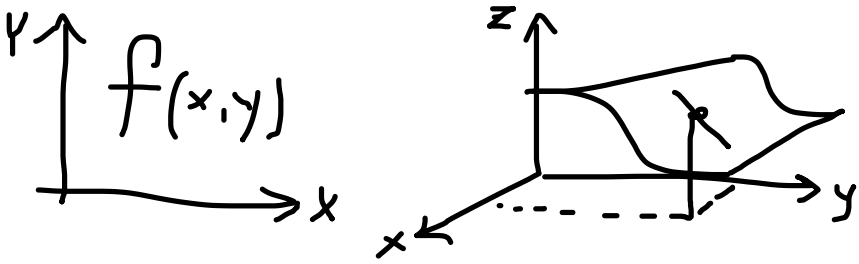
Grazie a questi grafici si possono confrontare gli esiti di diverse scelte dei parametri ε, α, γ.

Feature extraction

Si tratta di ricavare delle informazioni dall'immagine del tratto di strada che si sta percorrendo. Queste potrebbero essere facilmente desunte da cartelli stradali e/o da una linea di mezzeria. Tuttavia anche in assenza di queste facilitazioni, si possono ricavare informazioni sull'andamento dei bordi della strada analizzando l'immagine del tratto di strada visibile di fronte all'auto.

Tecnica HOG (Histogram of Oriented Gradient)

Si tratta di una tecnica per estrarre la direzione prevalente dei bordi di una figura
Si divide la figura (ridotta in scala di grigi) in blocchi 8x8 pixel e si calcola per ciascun blocco la frequenza degli angoli dei gradienti.
Il gradiente è la direzione di maggior pendenza di una superficie a due dimensioni



$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix}$$

Per calcolare Gx (derivata parziale di f rispetto a x) e Gy (derivata parziale di f rispetto a y) si possono usare le operazioni di convoluzione di Prewitt oppure di Sobel:

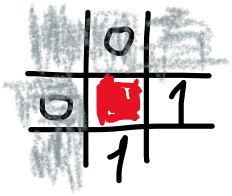
$$\begin{bmatrix} -1, 0, +1 \\ -1, 0, +1 \\ -1, 0, +1 \end{bmatrix}$$ Prewitt $$\begin{bmatrix} -1, 0, +1 \\ -1, 0, +2 \\ -1, 0, +1 \end{bmatrix}$$ Sobel

Data la semplicità dell'immagine della pista (bianco con bordo nero) si può utilizzare la più semplice matrice di calcolo

[-1, 0, +1] per la x

$$\begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix}$$ per la y

da applicare a ciascun pixel, dati i pixel del suo intorno:



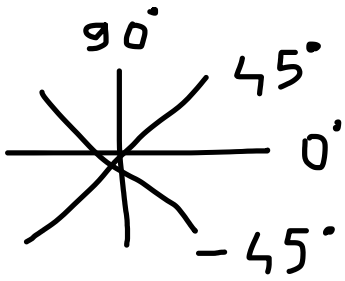
0 i pixel scuri del bordo, 1 i pixel bianchi della pista

Per il pixel centrale il gradiente è Gx = 1 – 0 = 1 e Gy = 0 – 1 = -1
L'angolo del gradiente si ottiene calcolando l'arcotangente a due parametri

θ = arctan2(Gy, Gx) = arctan2(-1, 1) = -45°

Ci interessa l'angolo che ha il bordo della pista nel punto (pixel) considerato: esso risulta ortogonale al gradiente. Pertanto basta aggiungere 90°, oppure togliere 90°, al valore di θ e si ottiene +45°

Nel blocco 8x8 si devono calcolare gli angoli per i 6x6 pixel centrali.
Si possono verificare i seguenti casi:
Gx = 1 Gy = 1 ➔ θ = 45° ➔ bordo a -45°
Gx = -1 Gy = -1 ➔ θ = -135° ➔ bordo a -45°
Gx = 1 Gy = -1 ➔ θ = -45° ➔ bordo a +45°
Gx = -1 Gy = 1 ➔ θ = -45° ➔ bordo a + 45°
Gx = 0 Gy = 1 ➔ θ = 90° ➔ bordo a 0° (orizzontale)
Gx = 0 Gy = -1 ➔ θ = -90° ➔ bordo a 0° (orizzontale)
Gx = 1 Gy = 0 ➔ θ = 0° ➔ bordo a 90° (a sinistra)
Gx = -1 Gy = 0 ➔ θ = -180° ➔ bordo a -90° (a destra)
Gx = 0 Gy = 0 ➔ θ = Non definito ➔ nessun bordo



Esempio: si consideri il seguente blocco 5x5, di cui si calcolano gli angoli per i pixel contrassegnati con una lettera

	A	B	C	
	D	E	F	
	G	H	I	
		auto		

A) Gx = 1, Gy = -1 ➔ bordo 45°
B) Gx = 0, Gy = -1 ➔ bordo 0°
C) Gx = 0, Gy = -1 ➔ bordo 0°
D) Gx = 1, Gy = 0 ➔ bordo 90°
E) Gx = 0, Gy = 0 ➔ Nessun bordo
F) Gx = 0, Gy = 0 ➔ Nessun bordo
G) Gx = 1, Gy = 0 ➔ bordo 90°
H) Gx = 0, Gy = 0 ➔ Nessun bordo
I) Gx = 0, Gy = 0 ➔ Nessun bordo

Si determina così un array di frequenze degli angoli che si ottengono:

(Gx, Gy)	angolo	Frequenza
(-1, 0)	-90	0
(1, -1) (-1, 1)	-45	0
(0, -1)	0	2
(1, 1) (-1, -1)	45	1
(1, 0)	90	2
(0, 0)	N	4
(0, 1)	180	

← Situazione impossibile

Questo array di frequenze consente di approssimare l'immagine vista dall'auto, in quanto ricava l'orientamento prevalente del bordo pista, se presente.
Nel caso in esame si capisce che c'è un bordo a sinistra e un bordo davanti che lasciano intendere la presenza di una curva a destra.

Invece nella seguente situazione

	A	B	C	
	D	E	F	
	G	H	I	
		auto		

(Gx, Gy)	angolo	Frequenza
(-1, 0)	-90	3
(1, -1) (-1, 1)	-45	0
(0, -1)	0	0
(1, 1) (-1, -1)	45	0
(1, 0)	90	3
(0, 0)	N	3

Da cui si deduce esservi un tratto di strada rettilineo, data la presenza di soli bordi a destra e a sinistra.

Dal punto di vista dell'auto, non interessa quello che c'è oltre il bordo pista, quindi, se ci si trova nei pressi della curva, la situazione potrebbe essere la seguente, dove ci sono solo due celle da considerare:

		A	B	
		auto		

Il corrispondente vettore delle frequenze è il seguente:

(Gx, Gy)	angolo	Frequenza
(-1, 0)	-90	0
(1, -1) (-1, 1)	-45	1
(0, -1)	0	1
(1, 1) (-1, -1)	45	0
(1, 0)	90	0
(0, 0)	N	0

Mentre un essere umano può ragionare sull'interpretazione di questi gradienti, il sistema automatico di apprendimento non effettua alcuna interpretazione degli stessi, ma si limita ad osservare l'esito delle diverse possibili azioni in corrispondenza degli stati in cui si trova l'auto.
Ne consegue l'individuazione dell'azione ottimale per ciascuno di questi stati.
Eventualmente, spetta poi all'essere umano analizzare la politica ottima di scelta dell'azione in corrispondenza di ciascuno stato e ricavare da questa la strategia di guida applicata dall'automobile.
Si ricorda che una importante caratteristica del reinforcement learning è la capacità di dedurre strategie di comportamento ottimale nelle diverse situazioni.

Approfondimenti futuri:

Approssimazione del dominio degli stati

Per ridurre le dimensioni dello spazio degli stati e di conseguenza ridurre i tempi di apprendimento

Clusterizzazione degli stati

The ***k*-medoids** problem is a clustering problem similar to k-means. The name was coined by Leonard Kaufman and Peter J. Rousseeuw with their PAM algorithm. Both the k-means and k-medoids algorithms are partitional (breaking the dataset up into groups) and attempt to minimize the distance between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the k-means algorithm, k-medoids chooses actual data points as centers (medoids or exemplars), and thereby allows for greater interpretability of the cluster centers than in k-means, where the center of a cluster is not necessarily one of the input data points (it is the average between the points in the cluster). Furthermore, k-medoids can be used with arbitrary dissimilarity measures, whereas k-means generally requires Euclidean distance for efficient solutions. Because k-medoids minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances, it is more robust to noise and outliers than k-means.

k-medoids is a classical partitioning technique of clustering that splits the data set of n objects into k clusters, where the number k of clusters assumed known *a priori* (which implies that the

programmer must specify k before the execution of a k-medoids algorithm). The "goodness" of the given value of k can be assessed with methods such as the silhouette method.

The medoid of a cluster is defined as the object in the cluster whose average dissimilarity to all the objects in the cluster is minimal, that is, it is a most centrally located point in the cluster.

In general, the k-medoids problem is NP-hard to solve exactly. As such, many heuristic solutions exist.

Partitioning Around Medoids (PAM)

PAM uses a greedy search which may not find the optimum solution, but it is faster than exhaustive search. It works as follows:

1. (BUILD) Initialize: greedily select *k* of the *n* data points as the medoids to minimize the cost
2. Associate each data point to the closest medoid.
3. (SWAP) While the cost of the configuration decreases:
    1. For each medoid *m*, and for each non-medoid data point *o*:
        1. Consider the swap of *m* and *o*, and compute the cost change
        2. If the cost change is the current best, remember this *m* and *o* combination
    2. Perform the best swap of m_{best} and o_{best}, if it decreases the cost function. Otherwise, the algorithm terminates.

The runtime complexity of the original PAM algorithm per iteration of (3) is $O(k (n-k)^2)$, by only computing the change in cost. A naive implementation recomputing the entire cost function every time will be in $O(n^2k^2)$. This runtime can be further reduced to $O(n^2)$, by splitting the cost change into three parts such that computations can be shared or avoided (FastPAM). The runtime can further be reduced by eagerly performing swaps (FasterPAM), at which point a random initialization becomes a viable alternative to BUILD.

Metrica: cosine similarity

The **cosine similarity** is a measure of similarity between two non-zero vectors defined in an n-dimensional space. Cosine similarity is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths. It follows that the cosine similarity does not depend on the magnitudes of the vectors, but only on their angle. The cosine similarity always belong to the interval [-1, +1]

For example two proportional vectors have a cosine similarity of 1, two orthogonal vectors have a similarity of 0, and two opposite vectors have a similarity of -1. In some contexts, the component values of the vectors cannot be negative, in which case the cosine similarity belongs to the interval [0, 1]

It is also used to measure cohesion within clusters in the field of data mining.