ieeta instituto de engenharia electrónica e telemática de aveiro

universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Machine Learning

LECTURE : Anomaly detection & Reinforcement Learning

Petia Georgieva
(petia@ua.pt)

Outline

1. **Anomaly detection**

2. **Reinforcement Learning**

   - **Markov Decision Process (MDP)**

   - **Value function, Value Iteration, Policy Iteration**

Part 1: Anomaly detection

Popular applications of anomaly detection

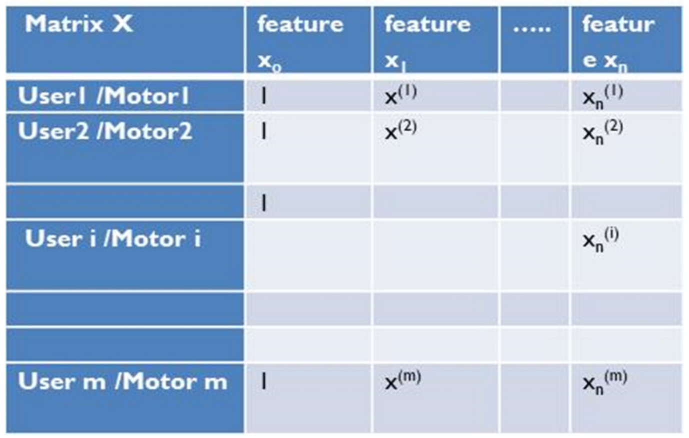
Fraud detection - $x^{(i)}$ – vector of features of user *i*

x_1 - how often does the user login

x_2 # of web pages visited / # of transactions

x_3 - the typing speed of the user

How to identify suspicious users ?



Matrix X	feature x_0	feature x_1	.....	featur e x_n
User1 /Motor1	1	$x^{(1)}$		$x_n^{(1)}$
User2 /Motor2	1	$x^{(2)}$		$x_n^{(2)}$
	1			
User i /Motor i				$x_n^{(i)}$
User m /Motor m	1	$x^{(m)}$		$x_n^{(m)}$

Manifacturing (e.g. aircaft engine features)

x_1= heat generated

x_2=vibration intensity,

x_3, x_4.. … other features

How to identify anomalous production (engines) for quality control ?

Monitoring computers in data center: $x^{(i)}$ = features of machine *i*

x1=memory use of computer

x2=number of disc accesses /sec

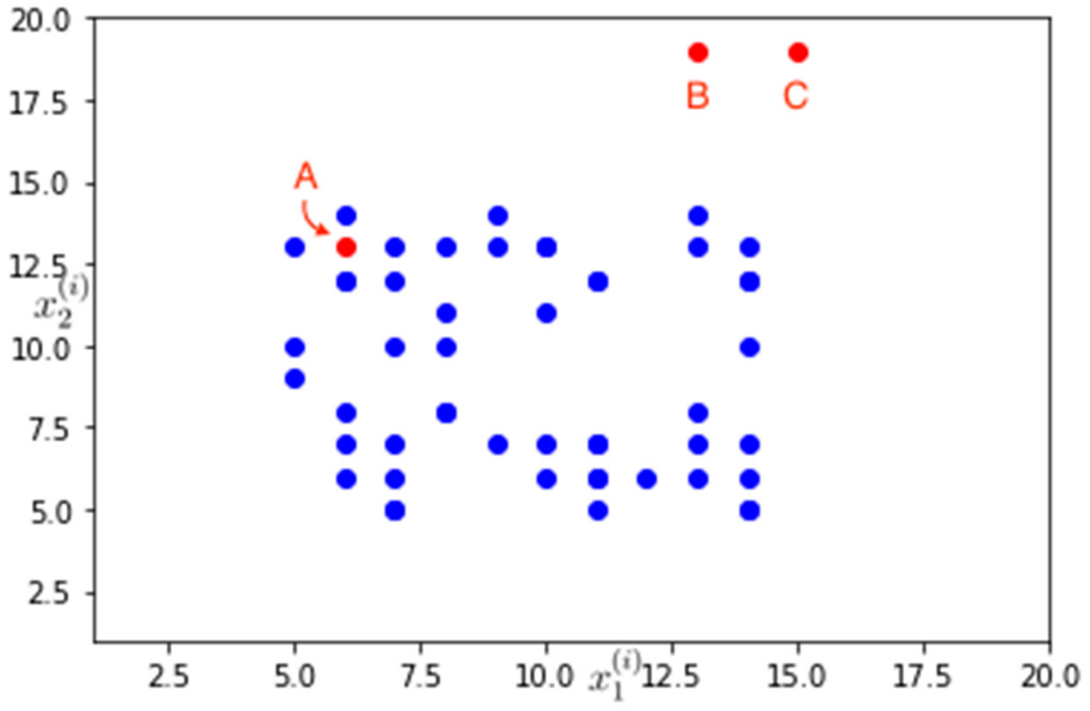
x3=CPU load

x4=network traffic &….other features…

How to identify if a computer is doing something strange and further inspect it?

Anomaly detection

We have a dataset of examples (blue points): $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$
Each example has two features x_1 and x_2.
We get new examples (red points): A, B, C
How to decide that A is not an outlier, B and C are anomalous (outliers) ?



Anomaly detection – How ?

From given regular (not anomalous) data get a model of what is considered as normal. For example – a probability model *p(x)* .

Identify anomalous case by checking if

$p(x_{test}) < \epsilon$ (flag as anomaly)

$p(x_{test}) >= \epsilon$ (flag as normal)

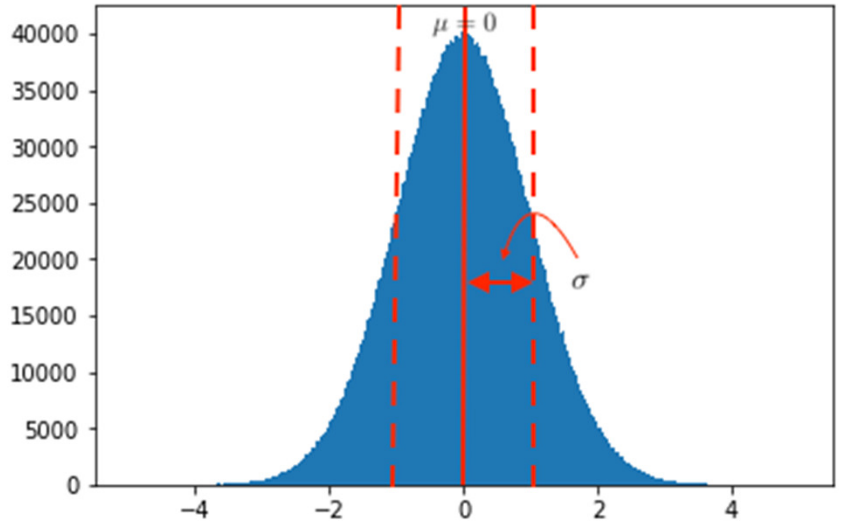
Gaussian (Normal) distribution

If $x \in \mathbb{R}$, and x follows Gaussian distribution with mean, μ and variance σ^2, denoted as,

$$x \sim \mathcal{N}(\mu, \sigma^2)$$

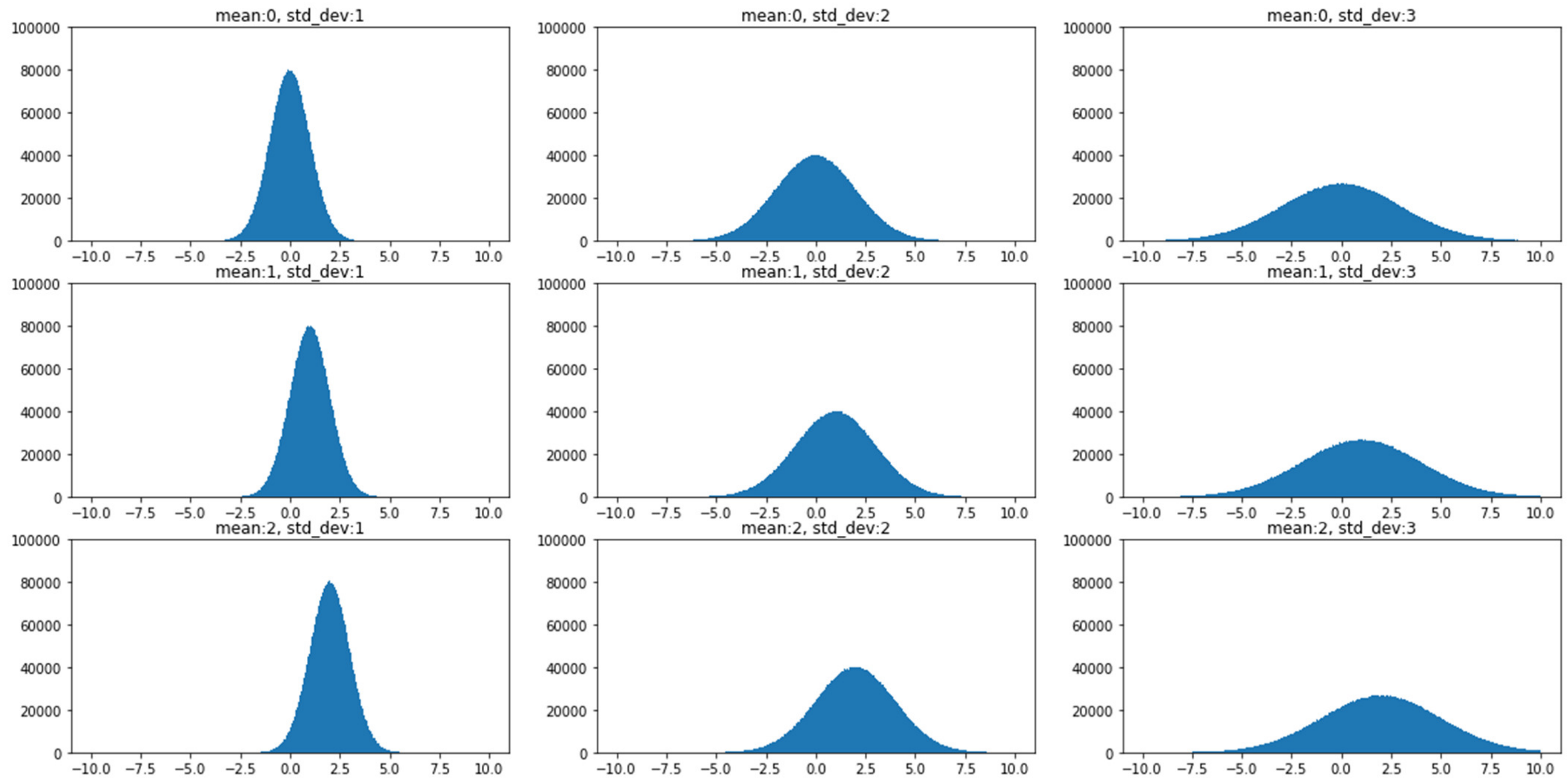
Standard normal Gaussian distribution (μ=0, standard deviation σ=1).
Density is higher around μ and reduces as distance from mean increases.
If we know parameters μ and σ, the probability of x in Gaussian distribution is:

$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$



Effect of μ and σ on Gaussian curve

Mean (μ) defines the centering of the distribution.
Standard deviation (σ) defines the spread of the Gaussian distribution.
As the spread increases the height of the plot decreases, because the total area under a probability distribution should be = 1.



Parameter estimation

Parameters (μ and σ) of the Gaussian distribution are estimated based on given data $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2$$

Remark: In statistics instead of 1/m is used 1/(m−1) , but in ML the above formulas are used. This change brings slightly different math properties but for high values of m both give similar result.

Density Estimation Algorithm

Given m training examples => $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$

Each example *i* has *n* features => $\{x_2^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}\}$

Major assumptions:
The features have Gaussian distribution and are independent.

$$
\begin{aligned}
x_1 &\sim \mathcal{N}(\mu_1, \sigma_1^2) \\
x_2 &\sim \mathcal{N}(\mu_2, \sigma_2^2) \\
&\vdots \\
x_j &\sim \mathcal{N}(\mu_j, \sigma_j^2) \\
&\vdots \\
x_n &\sim \mathcal{N}(\mu_n, \sigma_n^2)
\end{aligned}
$$

Compute μ and σ of each feature.

Compute *p(x_j)* of each feature.

Compute probability (density estimation) of all features *p(x):*

$$p(x) = p(x_1; \mu_1, \sigma_1^2)p(x_2; \mu_2, \sigma_2^2) \cdots p(x_n; \mu_n, \sigma_n^2)$$

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

Anomaly Detection Algorithm

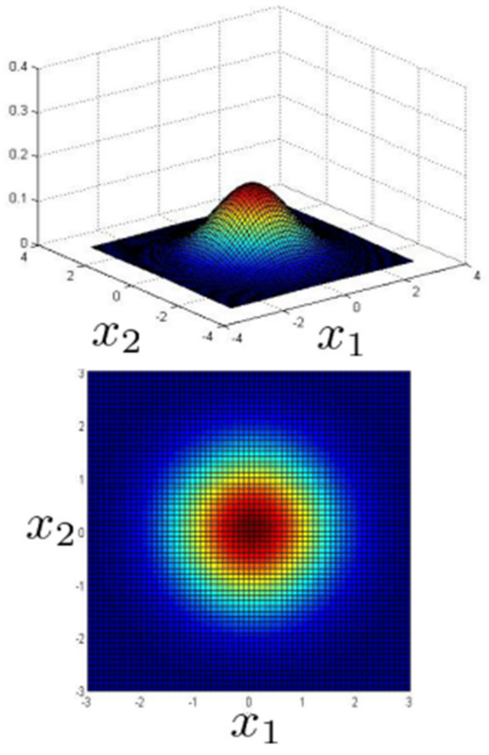
1. Choose features that you think might be indicative of anomalous examples.

2. Fit Gaussian distribution parameters (μ and σ) for each feature.

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

3. Given new example (x_{new}), compute $p(x_{new})$

4. Anomaly if $p(x_{new}) < \epsilon$ (some threshold)



Evaluation of Anomaly Detection System

When developing a learning algorithm making decisions is easier if we have a **single real-valued metric.**

Let we have some *labelled data*, of many normal examples and a few anomalous.

Train set (take ONLY normal examples !!!): $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$

Cross validation (CV) set (normal & anomalous exs.):

$$(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$$

Test set (normal & anomalous exs.):

$$(x_{test}^{(1)}, y_{test}^{(1)}), \ldots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$$

Ex. Manifacturing (e.g. aircaft engine features)
10000 good (normal) engines & 20 anomalous engines

Train set (only good examples): 6000 good engines (y=0)
CV set: 2000 good engines (y=0), 10 anomalous (y=1)
Test set: 2000 good engines (y=0), 10 anomalous (y=1)

Evaluation of Anomaly Detection System

1) Fit *p(x)* on train set (ONLY normal examples) $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$

2) Check the performance on cross-validation (CV) set.
From a range of possible values of ϵ, choose the best threshold (ϵ) to optimize some performance metric.

p(x_CV)< ϵ (anomaly)
p(x_CV)> ϵ (normal)

Possible performance metrics

- True positive, true negative, false positive, false negative
- Precision/Recall
- F1-score

Accuracy is not a good perf. measure because data is unbalanced (much more normal examples than anomalous).

3) Test the final model on test set.

p(x_test)< ϵ (anomaly)
p(x_test)> ϵ (normal)

Anomaly Detection vs. Supervised Learning

- Very small number of positive (anomalous) examples.
- Large number of negative (normal) examples.
- Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like.
- Future anomalies may look nothing like any of the anomalous examples seen before.

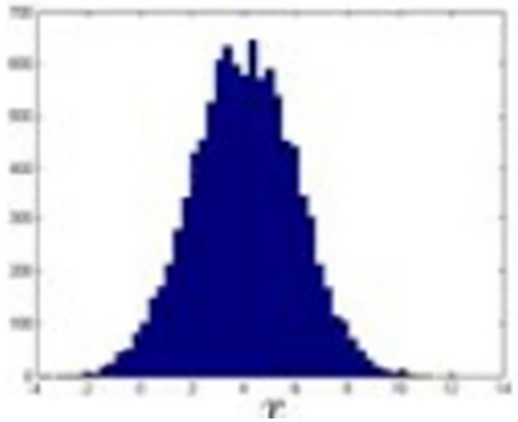
Exs.: fraud detection, monitoring computers in data centers, suspicious computer;

- Large number of both positive and negative examples
- Enough positive examples for the algorithm to get a sense of what positive examples are like.
- Future positive examples likely to be similar to ones in training set.

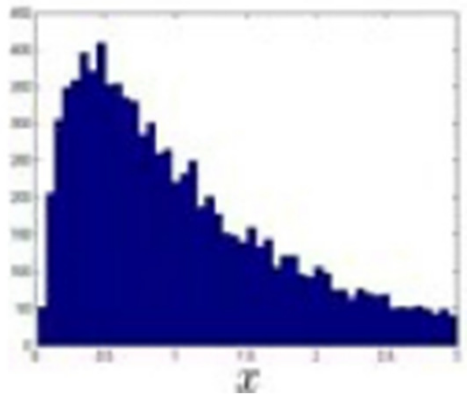
Exs.: Spam/fishing email classification; cancer prediciton/classification

Choosing Features

If features have Gaussian distribution =>
apply the discused anomaly detection algorithm .



If features do not have Gaussian distribution =>
play with different transformations of data to get more Gaussian curves.



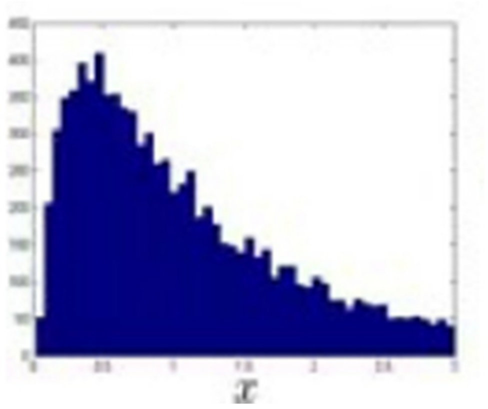
universidade de aveiro

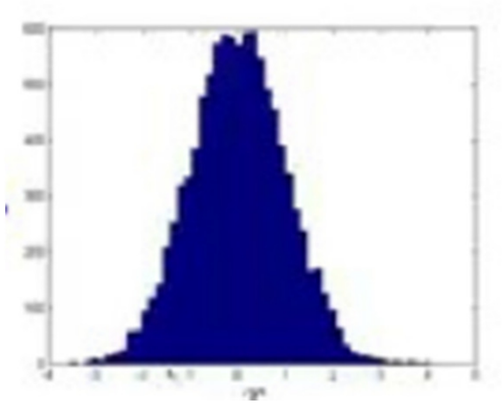
Choosing Features

Popular feature transformations =>

- $log(x)$
- $log(x + c)$
- $\sqrt{x}$

for example: x => log(x) => much more Gaussian curve





Error Analysis for Anomaly Detection

Want: p(x) large for normal examples x;
p(x) small for anomalous examples x.

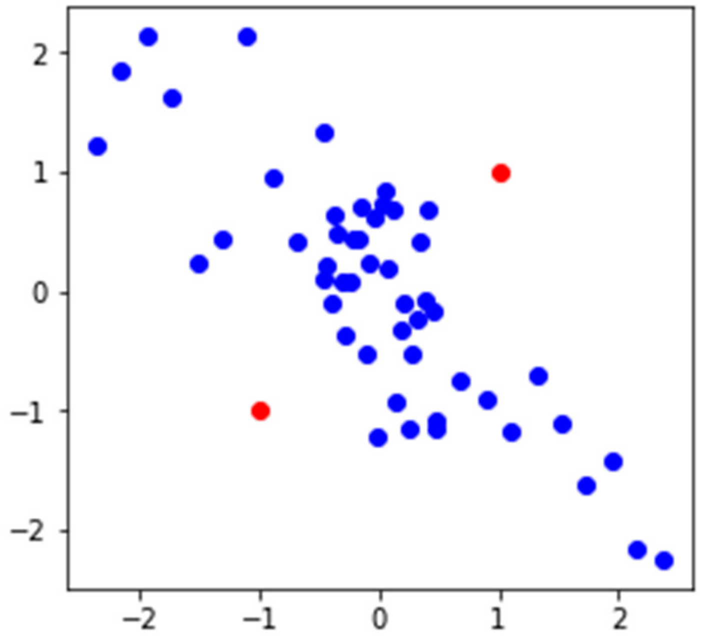
Most common problem:
p(x) is comparable (say both large) for normal and anomalous examples.

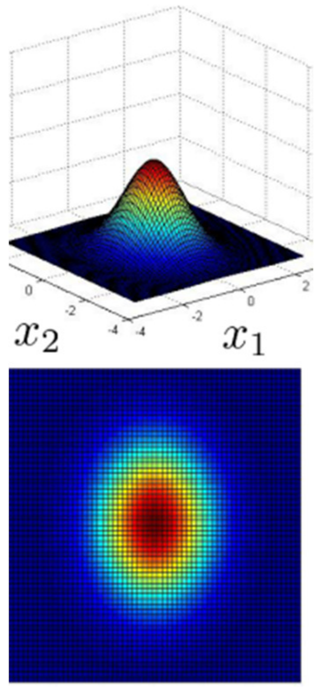
Solution: Make error analysis to create new features
Look at the anomalies the algorithm did not flag correctly (the mistakes) and try to create some new feature that may take unusually different (large or small) values in the event of anomaly and thus distinguish the abnormal ex.

Ex. Monitoring computers in data center
x1=memory use of computer
x2=number of disc accesses /sec
x3=CPU load
x4=network traffic

Feature engineering (new features)
x5=CPU load /network traffic
x6= (CPU load)^2 /network traffic

Multivariate Gaussian Distribution

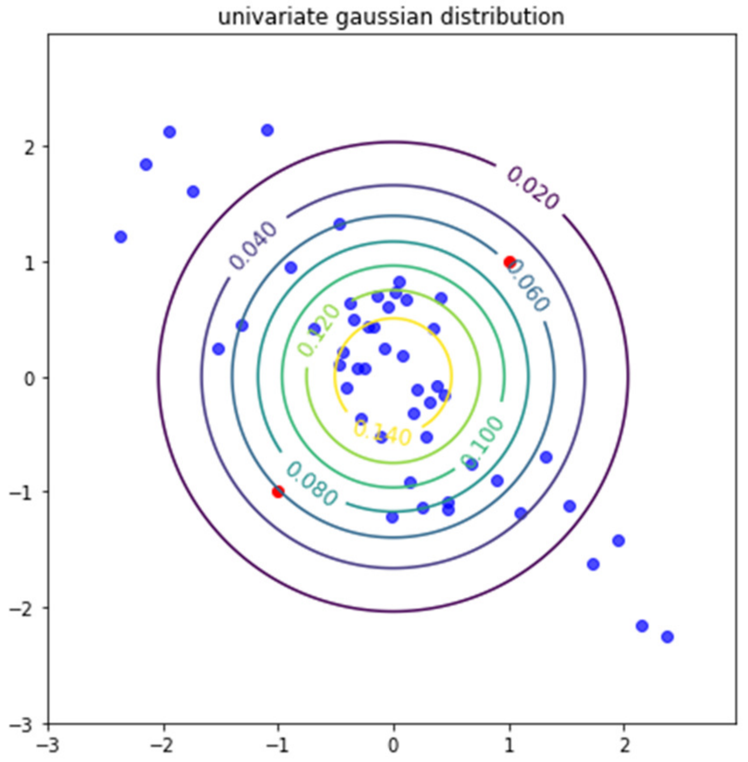


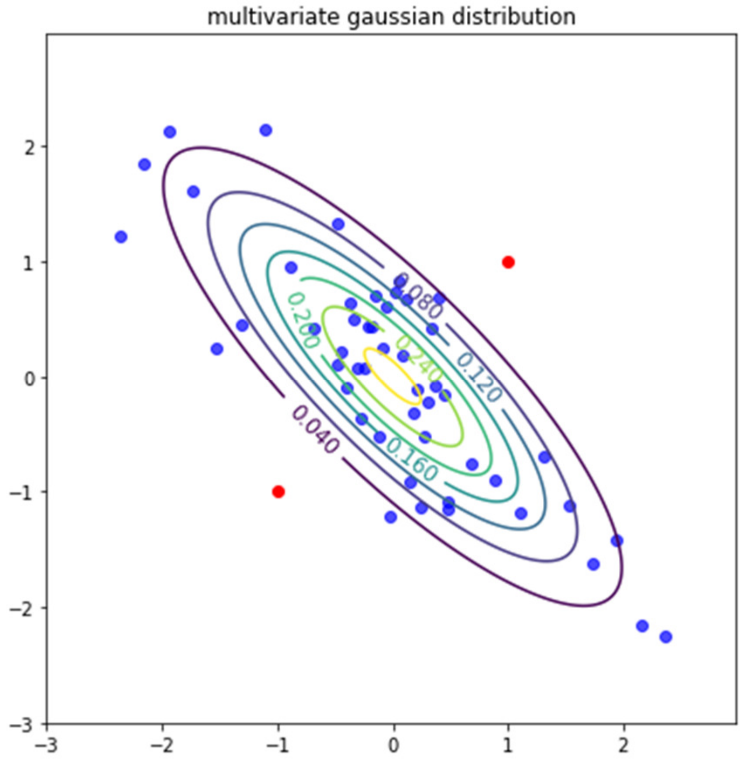


If we use univariate Gaussian distribution for this data, the contour plots (fixed value of the probability) will be circles (if both variances are the same) or axes alined elipces (if the variances are different) . The red points will have relatively high probability => not flaged as outliers.

But features x_1 and x_2 are negatively correlated (one increases, the other decreases). The assumption of independance is violated.
Red points are outliers.

Univariate vs.Multivariate Gaussian Distribution





Univariate Gaussian distribution considers separately probability models for each p(x_1), p(x_2) => it will not flag the red points as outliers.
Better use Multivariate Gaussian distribution.

Multivariate density estimation

Given training data, estimate μ (nx1 vector) and **Σ (nxn covariance matrix):**

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

For a new example x, compute:

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}\,|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

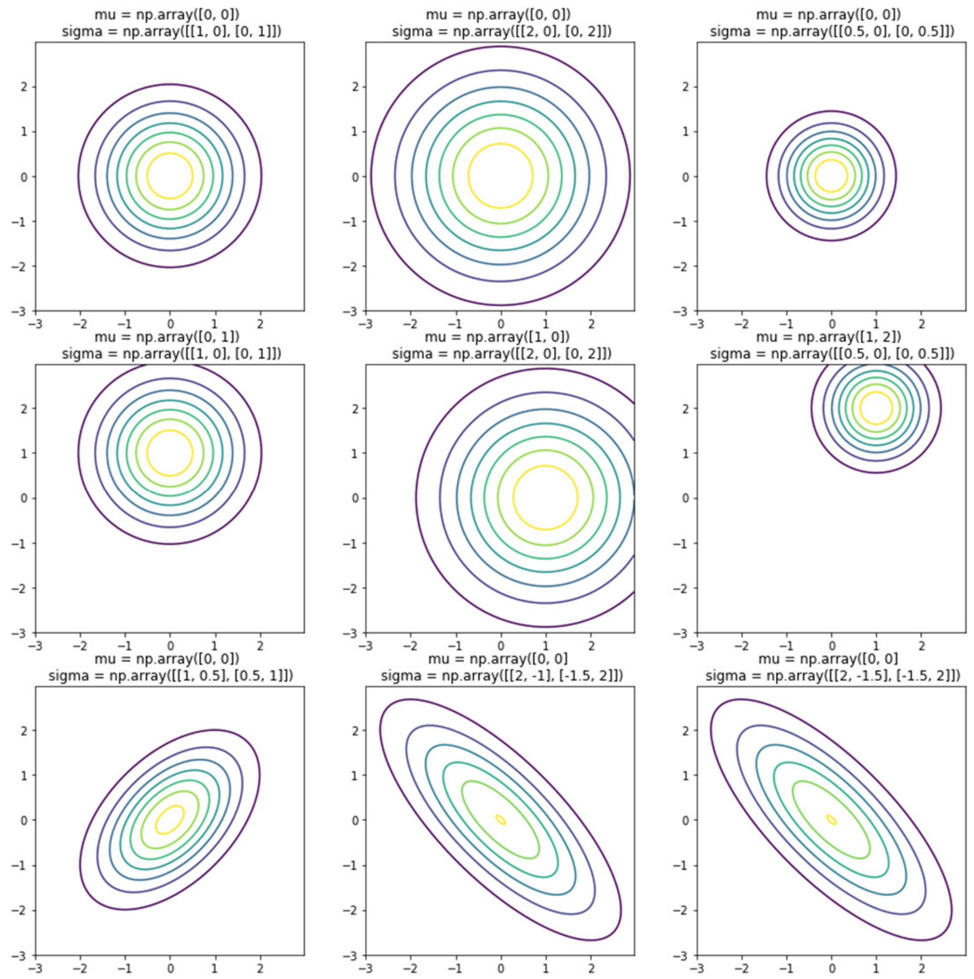
Flag as anomaly if $p(x)<\epsilon$.

Σ – nxn covariance matrix
|Σ| - determinant of matrix Σ.
Σ - symmetric about the main diagonal.

Σ - major difference between univariate and multivariate Gaussian !!!

universidade de aveiro

Effect of Mean and Covariance Matrix Shifting



μ shifts the center of the distribution.

Diagonal elements of Σ vary the spread of the distribution along the corresponding features

Off-diagonal elements of Σ (sigma) show the correlation among the features:

Positive off-diagonal values of Σ => positive correlation

Negative off-diagonal values of Σ => negative correlation

Univariate Gaussian distribution is a special case when off-diagonal values of Σ are 0.

Original vs. Multivariate Gaussian models for Anomaly detection

Original model

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)^2$$

$$p(x) = p(x_1;\mu_1,\sigma_1^2)p(x_2;\mu_2,\sigma_2^2)\cdots p(x_n;\mu_n,\sigma_n^2)$$

Manually create features to capture anomalies
(e.g. x_new=CPU load /network traffic)

Computationally cheaper, scales better to large number of features (n)

OK even if the training set size (m) is small

Multivariate Gaussian

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}\,|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Automatically captures correlations between features

Computationally more expensive, takes time to compute inverse of Σ if number of features (n) is large

Must have training set size (m) >> number of features (n) , otherwise Σ is singular and not invertible.
In practice m>10*n

Part 2: Reinforcement Learning

Machine Learning Approaches

Supervised Learning

• Training data: (x, y) (features, label) samples. We want to predict y to minimize a loss function.

• Regression, classification

Unsupervised Learning

• Training data: x (features) samples only. We want to find “similar” points in the x space.

• Clustering

Reinforcement Learning

• Training data: (s, a, r) (state, action, reward) samples. We want to find the best sequence of decisions so as to maximize long-term reward.

• Robotics

Reinforcement Learning - Online Learning

In Reinforcement Learning (RL), we do not give the “right answer “ (as with the supervised learning) but instead provide a **reward**, which indicates to the learning machine (called AI agent) when it is doing well, and when it is doing poorly.

Reinforcement learning can be applied to many different areas.

- Robotics: in which direction and how fast should a robot arm move?
- Mobility: where should taxis go to pick up passengers?
- Transportation: when should traffic lights turn green?
- Recommendations: which news stories will users click on?
- Network configuration: which parameter settings lead to the best allocation of resources?

Markov Decision Process (MDP)

MDP is the formalism in which RL problems are usually defined.

MDP comprises of the following tuple ***(S, A, P, R, γ)***:

S – set of **states** (e.g. in autonomous helicopter flight, S may be the set of 3D position (x,y,z) and velocities (dx, dy, dz) of the helicopter)

A –set of **actions** (e.g. the set of possible directions (North, South, East, West) in which one can push the helicopter's control stick)

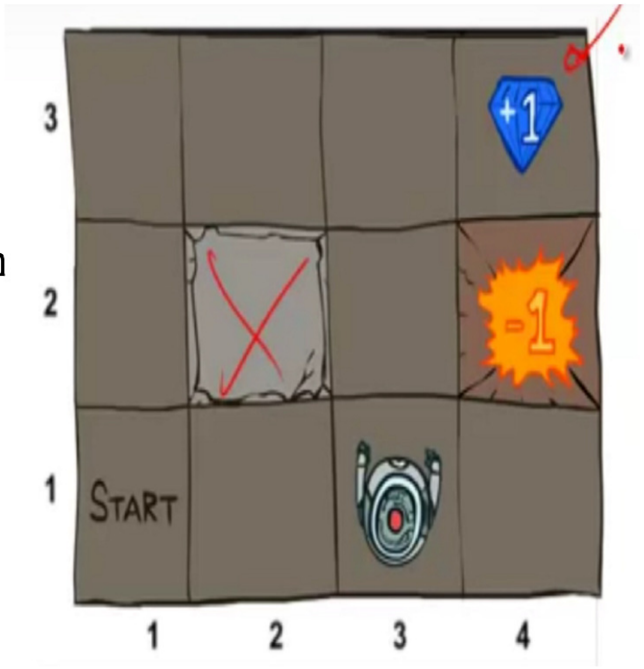
P(s,a,s´) - state **transition probabilities**. What is the probability being in state $s \in S$ and applying action $a \in A$ to go to a new state $s´ \in S$.

R (s,a) – **reward function** (of the state *s* and/or the action *a*)

$\gamma \in [0,1)$ – **discount factor** (trade off between current & future rewards)

Example: Robot navigation problem

Robot lives in a grid world.
There are obstacles (X) on its path.
Goal: go to the diamond; don't die in the fire.



Robot MDP:
States: sells are states (11 states) -robot position
Actions: go {North, South, East, West}

If we command the robot to go North, due to its noisy dynamics it may go 80%North; 10%East; 10%West – probabilistic world.
Transition probabilities:
P(s(1,3), North, s'(2,3))=0.8
P(s(1,3), North, s'(1,4))=0.1
P(s(1,3), North, s'(1,2))=0.1
All other state transition probabilities =0

Rewards:
Terminal states: R(s(3,4))=+1; R(s(2,4))=-1
For all other states R(s)=-0.02 (cost of state, e.g. charge for resourse consumption)

Dynamics of Markov Decision Process (MDP)

Start in some state s_0, choose some action $a_0 \in A$ and transit to some next state s_1, with probability $P(s_0,a_0,s_1)$. Then, pick another action a_1 and transit to some other state s_2 with probability $P(s_1,a_1,s_2)$. Then pick a_2, and so on. . . .

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots$$

Total reward (total payoff): $R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots$

$\gamma < 1$, the weigth of wins or losses in the future is less than today

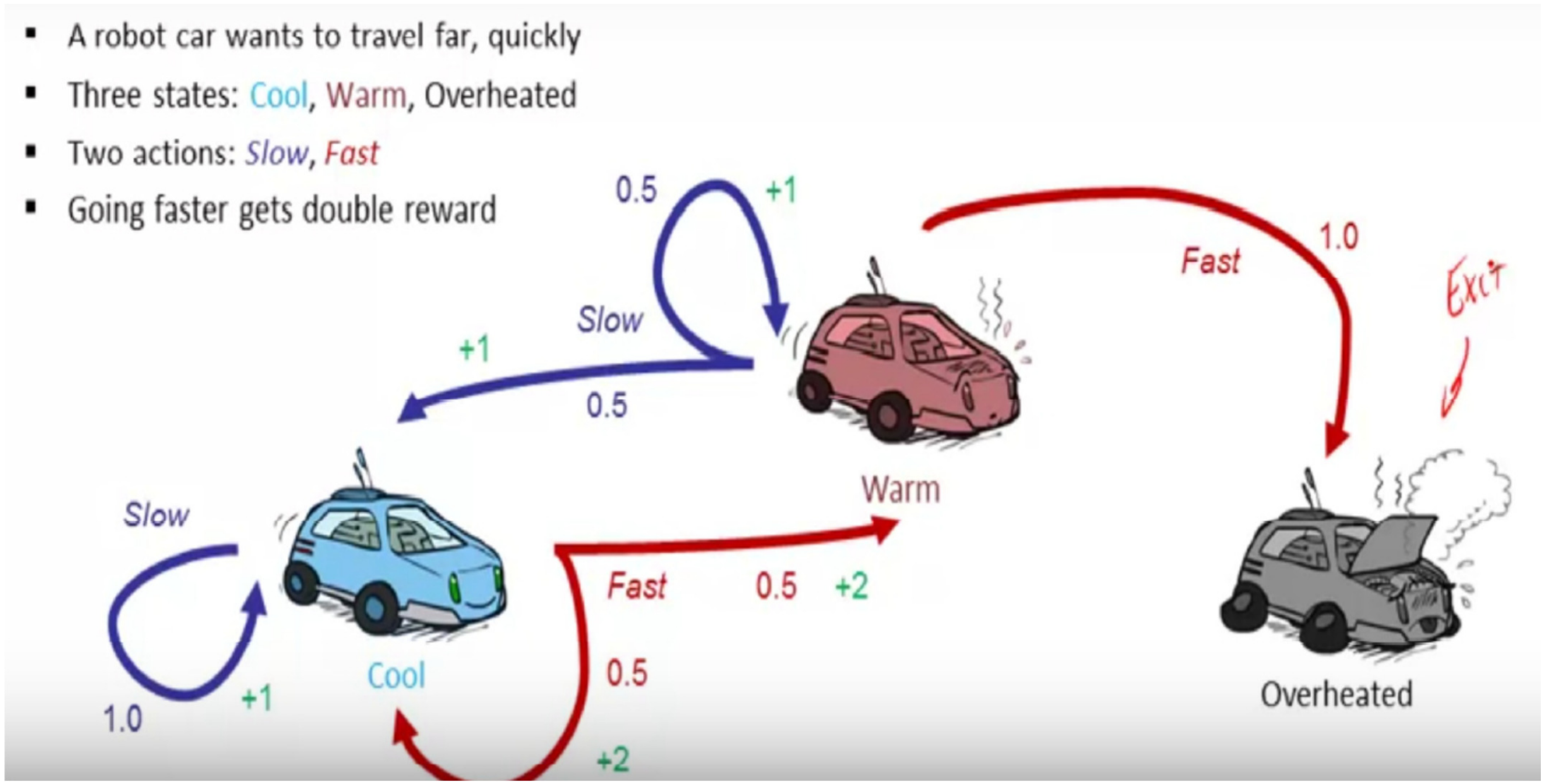
Total reward as a function of the states only:

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

Goal: Choose actions over time $(a_0, a_1, a_2 \ldots)$ to maximize the expected value E[.] of the total reward (total payoff):

$$\max \mathrm{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots]$$

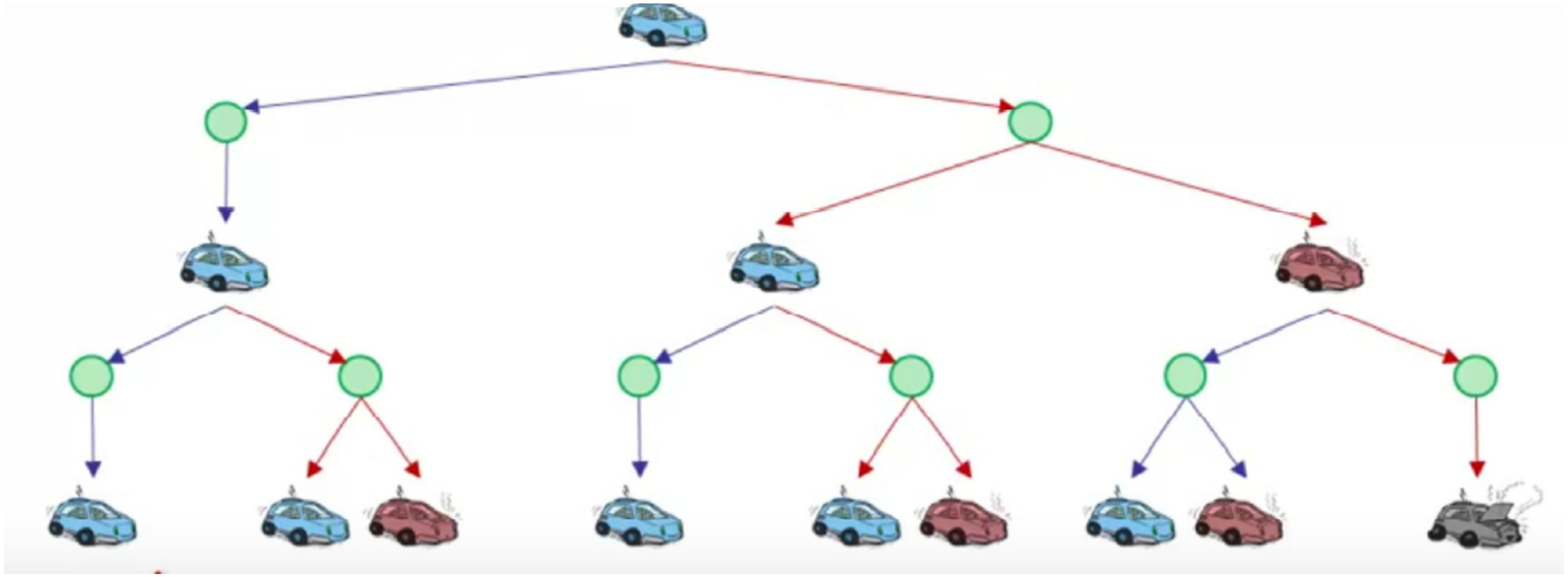
Example: Car driving



- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward

Example: Car driving -

3 dimensional state transition probability matrix *P(s,a,s')*



Transition probability between states for Action: Drive Slow (Reward =1) : P(s, 'drive slow', s´)			
	Cool	Warm	Overheated
Cool	1	0	0
Warm	0.5	0.5	0
Overheated	0	0	1

Transition probability between states for Action: Drive Fast (Reward=2) : P(s, 'drive fast', s´)			
	Cool	Warm	Overheated
Cool	0.5	0.5	0
Warm	0	0	1
Overheated	0	0	1

MDP QUANTITIES

Policy (π): A policy tells us which action to take, given the current state (mapping from states to actions S->A) .

State Value function: is the expected total payoff starting in state *s* and taking actions according to one chosen policy π (mapping from state to real number)

$$V^{\pi}(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \,\middle|\, s_0 = s, \pi\right]$$

Optimal State Value function (the best expected total payoff over all possible action policies)

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

Variant of State Value function (Bellman equations) – key eq. for solving MDP

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

Optimal value function (Bellman equations)

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

Optimal Policy (optimal action from state *s*) gives the action for which the value function is optimal

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

Value iteration

Value iteration – algorithm 1 for solving MDPs with finite-state and finate action spaces. Compute for each state and all possible actions at this state ((usually used in practice).

```
1. For each state $s$, initialize $V(s) := 0$.

2. Repeat until convergence {

       For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s')V(s')$

   }
```

Value function iteration (car driving)

	V(cool)	V(warm)	V(overheated)
t_0	0	0	-10
t_1	2 (a=fast)	1 (a=slow)	-10
t_2	3.5(a=fast)	2.5(a=slow)	-10
t_3	5(a=fast)	4 (a=slow)	-10
…			

V(cool) =[0 2 3.5 5] – value of state *cool* if driving *fast* increases
V(warm) =[0 1 2.5 4] - value of state *warm* if driving *slow* increases

Policy iteration

Policy iteration – algorithm 2 for solving MDPs with finite-state and finate action spaces. Compute for each action and all possible states. At step b) a linear system of eqs. for all states is computed. Recommeded when the states are not too many.

1. Initialize π randomly.
2. Repeat until convergence {
   (a) Let $V := V^{\pi}$.
   (b) For each state s, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s')V(s')$

   }

Value function iteration (car driving)			
	V(cool)	V(warm)	V(overheated)
t_0	0	0	-10
t_1	2 (a=fast)	1 (a=slow)	-10
t_2	3.5(a=fast)	2.5(a=slow)	-10
t_3	5(a=fast)	4 (a=slow)	-10
…			
.			

Policy: If in state *cool* , the best policy (the best reward) is to drive *fast*
If in state *warm* the best policy (the best reward) is to drive *slow*

Learning a model for MDP

MDP model *(S, A, P, R, γ)*

Usually S (states), A (actions) and γ (discount) are known but the state transition probabilities P and the rewards R are not given.
If we do not know P and R => estimate them from experience (data)

$$
\begin{aligned}
&s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} s_3^{(1)} \xrightarrow{a_3^{(1)}} \dots \\
&s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} s_3^{(2)} \xrightarrow{a_3^{(2)}} \dots \\
&\dots
\end{aligned}
$$

$$P(s,a,s') = \frac{\#\,\text{times take action a in state s and got to s}'}{\#\,\text{times take action a in state s}}$$

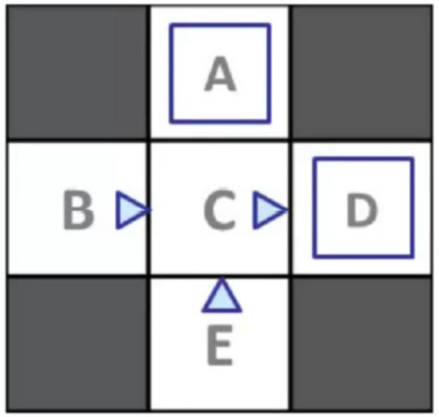
If some action *a* was never taken in state *s*, estimate transition probability to be the uniform distribution over all states

$$P(s,a,s') = \frac{1}{\#\,\text{of states}}$$

For **R – similar procedure.**
Estimate the reward R(s) in state *s* to be the average reward observed in *s*.

Learning a model for MDP - example

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

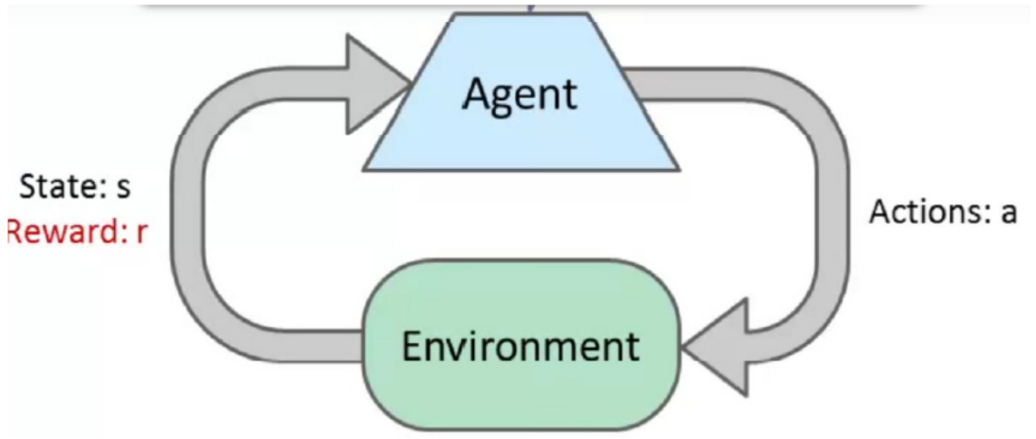
$\hat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
...

$\hat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
...

Reinforcement Learning - concept



Basic idea: Sequential decision making (on-line learning)

Agent takes actions.

Receive feedback from the environment in the form of rewards.

Learn to act so as to maximize the rewards.

Learning is based on past episodes.

Continuous state MDPs

So far, we considered MDPs with a finite number of states (e.g. cool, warm, overheated)

But MDPs may have an infinite number of states (states with infinite possible values)

Examples:

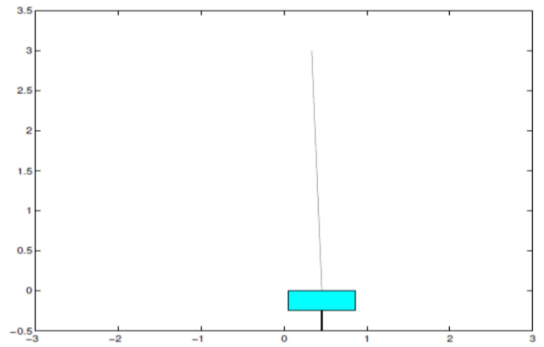
1. Car states : $(x, y, \theta, dx/dt, dy/dt, d\theta/dt)$ - 6 states
position (x, y); orientation θ; velocity in x and y directions $dx/dt, dy/dt$; angular velocity $d\theta/dt$.
There is an infinite number of possible positions and orientations for the car.

2. Inverted pendulum states: $(x, \theta, dx/dt, d\theta/dt)$, θ is the angle of the pole – 4 states

3. Helicopter flying in 3d space has states of the form $(x, y, z, \varphi, \theta, \psi, dx/dt, dy/dt, dz/dt, d\varphi/dt, d\theta/dt, d\psi/dt)$, where the roll φ, pitch θ, and yaw ψ angles specify the 3d orientation of the helicopter. 12 states

Continuous state MDPs - discretization

Example: Inverted pendulum
State of the cart and pole has 4 variables:

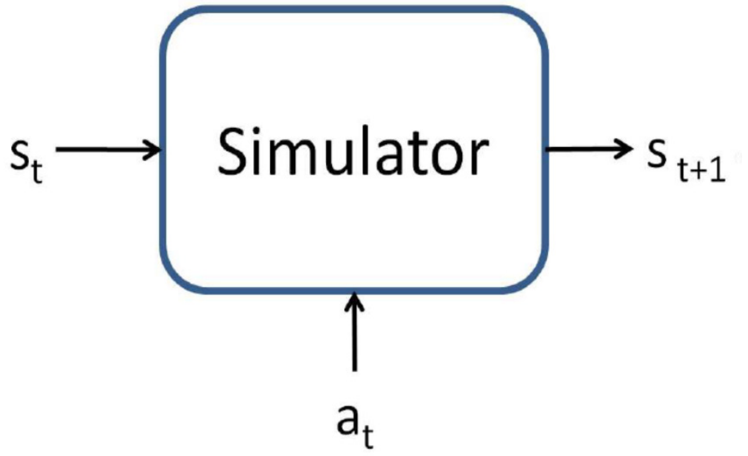


- cart position (x),
- cart velocity (dx/dt)
- angle of the pole (θ,)
- angular velocity of the pole ($d\theta/dt$)

Solution: Continuous state vector (x, θ, dx/dt, $d\theta/dt$) mapped into a grid of discrete states. Apply value iteration or policy iteration to solve for the best policy in the discretized MDP.

Curse of dimensionality – if state has 10 variables and each variable is discretized into 100 values => 100^{10} discrete states !!!

MDP physical model or simulator



Simulator of the environment obtained by using the laws of physics to calculate what position and orientation the object will be in time *t+1*, given the current state at time *t* and the action *a* taken, assuming the system parameters are given (e.g. pole length, pole mass, etc.)

MDP model from real data

Execute m trials in which take a sequence of actions in T time-steps. This can be done picking actions at random or executing a specific policy (e.g. always North), or via some other rules for choosing the actions. Observe m state sequences like the following:

$$\begin{array}{l}
s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} \cdots \xrightarrow{a_{T-1}^{(1)}} s_T^{(1)} \\
s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} \cdots \xrightarrow{a_{T-1}^{(2)}} s_T^{(2)} \\
\cdots \\
s_0^{(m)} \xrightarrow{a_0^{(m)}} s_1^{(m)} \xrightarrow{a_1^{(m)}} s_2^{(m)} \xrightarrow{a_2^{(m)}} \cdots \xrightarrow{a_{T-1}^{(m)}} s_T^{(m)}
\end{array}$$

Use data to learn for example a linear (regression) model of the form

$$s_{t+1} = As_t + Ba_t \quad \Rightarrow \quad \arg\min_{A,B} \sum_{i=1}^{m} \sum_{t=0}^{T-1} \left\| s_{t+1}^{(i)} - \left(As_t^{(i)} + Ba_t^{(i)} \right) \right\|^2$$

Implementing Reinforcement Learning

- **“Offline” version:** we have access to several state-action trajectories that form our training data.

- **“Pre-training” version:** we have access to a simulation of the environment that tells us what the state will be when we take an action.
We will train the RL algorithm on the simulator before deploying it in the real world.

- **“Fully online” version:** we can only learn about our environment by directly interacting with it.

The “pre-training” version is often used in practice, as it limits the risk of taking bad actions in deployment without requiring extensive training data.