Unsupervised Learning -

Clustering -

A clustering algorithm looks at a number of data points and automatically finds data points that are related or similar to each other.

Grouping unlabeled examples.

K-Means Algorithm

Step 0: Take 2 random points (centroid), to which can be the centre of the 2 clusters

Step 1: Assign each point to its closest centroid

Step 2: Recompute the centroid (find the average of all points that where assigned to it)

Goto Step 1

(If no further changes occur, stop algorithm)

Note: No. of centroids are K for K clusters.

Points are denoted as $x^{(1)}, x^{(2)}, x^{(3)}, \ldots$

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$

Repeat {

assign points to cluster centroids

for $i = 1$ to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

move cluster centroid.

for $k = 1$ to K

μ_k = average (mean) of points assigned to cluster k

}

K-Means optimization objective.

$c^{(i)}$ → index of cluster $(1, 2, \ldots, K)$ to which example $x^{(i)}$ is currently assigned

μ_k = ~~centroid cluste~~ cluster centroid k

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

Cost function = $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \mu_{c^{(i)}} \|^2$

$= \frac{1}{m} \sum_{i=1}^{m} \underbrace{\| x^{(i)} - \mu_{c^{(i)}} \|^2}_{\text{Distortion function}}$

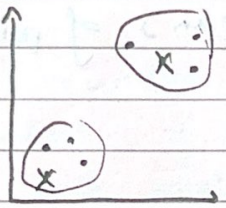
Initializing K-Means

↳ Random Initialization

Choose $K < m$

Randomly pick K training examples

Set $\mu_1, \ldots, \mu_K$ equal to these K examples



This method can create different clusters & all these sets of clusters may not be optimum.

Thus, we run it n times & find cost function (J) of all sets & choose the one with minimum J

For $i = 1$ to 100 {

Randomly initialize K-means.

Run K-means. Get $c^{(1)}, \ldots, c^{(m)}, \mu_1, \mu_1, \ldots \mu_K$

Compute cost function (distortion)

$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \mu_1, \mu_K)$

}

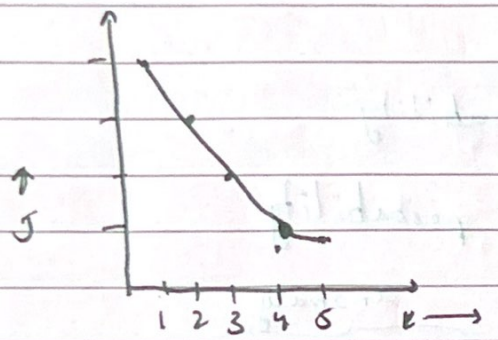
Pick set of clusters that gave lowest cost

Choosing Number of clusters

~~Elbow method~~ → ~~drawing a graph~~

Elbow method → drawing a graph ~~against~~ of J against K.

Where a sudden change occurs we take that as K



∴ K = 4

because J decreases slowly after K=4

Best method to choose K is to see what we use the K value depending on how we use the clusters

(3 of S sizes for shirts)

↑ S, M, L ↑ XS, S, M, L, XL

→ common use → Fraud Detection
→ manufacturing

Anomaly Detection

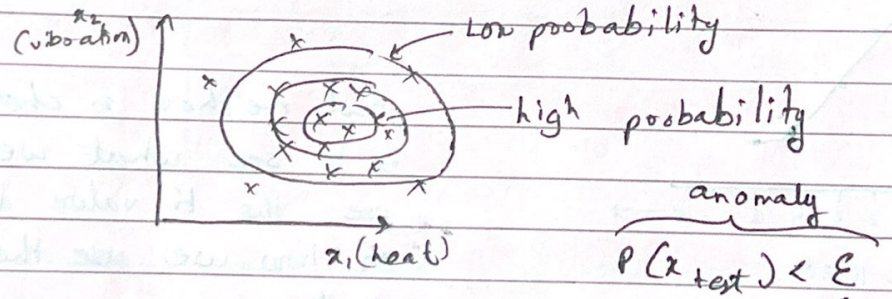
Basically when a new input of data differs from the usual pattern.

→ Density Estimation

Given a training set of m examples, we first train a model for probability of x being in dataset

Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

Model $p(x)$:



anomaly

$p(x_{test}) < \varepsilon$

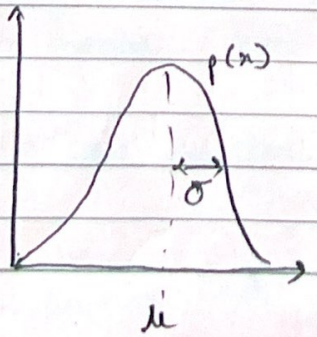
↑ small number

Gaussian (Normal) Distribution

Say x is a number

Probability of x is determined by a Gaussian with mean μ, variance σ^2

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



Parameter Estimation for normal distribution

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

Anomaly Detection Algorithm

Training set: $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}, \ldots, \vec{x}^{(m)}\}$

Each example $\vec{x}^{(i)}$ has n features

represented as x_n

$$\therefore p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * p(x_3; \mu_3, \sigma_3^2)$$

$$* \_\_\_\_\_ * p(x_n; \mu_n, \sigma_n^2)$$

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

where

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \qquad \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

Evaluation in Anomaly Detection System.

Reinforcement Learning

Type of ML that makes decisions by performing actions in an environment to achieve a goal. The agent learns through trial & error.

Reward - correct action - +1

Penalty - negative action - -100

Basic flow

- Code starts at state 's'
- Takes an action 'a'
- Gets a reward 'R(s)'
- Reaches new state 's''

Return in RL

(If you have 5 dollars infront of you & 10 $ at a distance 30 min away, which one will you choose?)

Return essentially helps us decide which award has a better value depending on convenience

In simpler terms, ~~reward~~ return is a culmination of all rewards

Discount factor is a small number (<1) which is added to the return statement which helps it prioritize quick gains

$\therefore$ Return $\Rightarrow R_1 + \gamma(R_2) + \gamma^2(R_3) + \gamma^3(R_4) + \gamma^4(R_5) + \cdots$

γ = discount factor

Policy
set of rules that defines the actions an agent takes to maximize its return

state $\xrightarrow[\pi]{\text{policy}}$ action

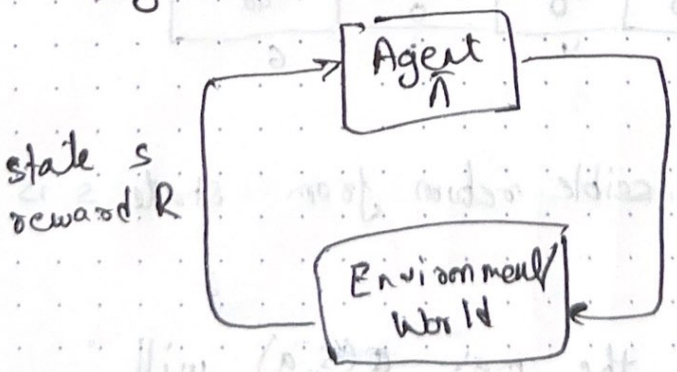
$s \rightarrow a$

A policy is a function $\pi(s) = a$ mapping from state to actions. Tells us what action to take for given state s

States → representation of possible conditions agent is in
Reward → feedback from environment
Action → Action is a decision to move

Markov Decision Process (MDP)
(formalization of all things based before)

(only current state matters for decision making)

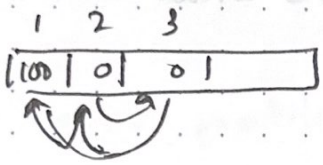


State Action value function (Q-function)

(allows us to calculate max return for a state)

$Q(s, a)$ = Return if you
- start in state s
- take action a (once)
- then behave optimally after that

Ex $Q(2, \rightarrow)$ = go to new state k then behave optimaly to reach desired target

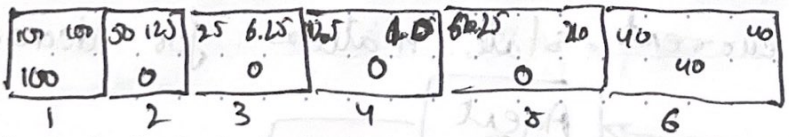


If $\gamma = 0.5$ then $Q(2, \rightarrow) = 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$

$= 12.5$

Ex $Q(2, \leftarrow) = 50$

So ~~if we take an example~~



The best possible return from state s is $\max_a Q(s, a)$

$\therefore$ At a state the max $Q(s, a)$ will tell us the action 'a' to take.

~~Q function Example~~

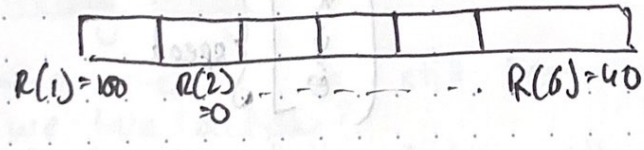
Bellman Equation

Helps compute the Q-Function (Breaks down Q-factor into a recursive problem)

We know that

$Q(s,a)$ = Return if you
- start in state s
- take action a (once)
- then behave optimally after that

$s \rightarrow$ current state
$R(s) \rightarrow$ reward of current state
$a \rightarrow$ current action
$s' \rightarrow$ state you get to after action a
$a' \rightarrow$ action that you take in state s'



$\therefore$ Bellman Equation $\Rightarrow Q(s,a) = R(s) + (\gamma * \max_{a'} Q(s',a'))$

Note: $\max_{a'}$ max over all possible actions

~~Random (Stochastic) Environment~~ 0.5 → 0.25(2.5)

Continuous State Spaces

In previous examples, we depicted distinct states, but in actual applications, these states are actually continuous.

Ex - a self driving car will be constantly on the move
so the state will include its speed in X or Y axis, angle (θ), ~~etc~~ position in X or Y axis, etc

∴ we can denote state as

$$s = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \Big\} \text{speed}$$

State Value function for continuous state space

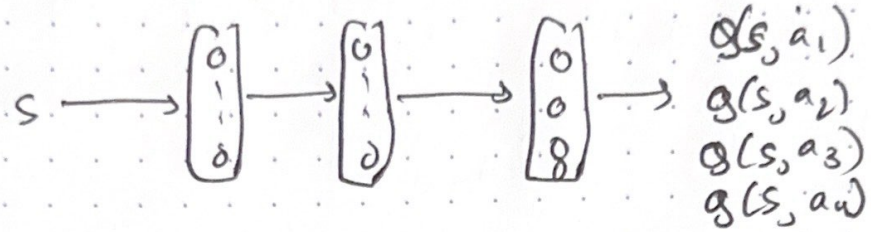
s → current state
a → action

$$\vec{x} = \begin{bmatrix} s \\ a \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \end{bmatrix} \rightarrow Q(s,a) = y$$

We need to create a light NN to help compute $Q(s,a)$

Note: if there are multiple actions, use binary encoding to specify action

A way to improve the Deep NN for more efficiency is to output all 'a' outputs.



So instead of taking inferance of all possible actions in a state, we can simply pass the state & get output for all actions

ε-Greedy Policy

When the RL is still learning, how do we take action?
ε-greedy Policy helps with this

~~In~~ some state s

- With probability 0.95, pick the action a that maximises ~~Q(s,a)~~ Q(s,a)
- With probability 0.05, pick an action a randomely

ε = 0.05 ✱

During training the NN may get stuck and not consider one option, so the 0.05 chance to choose a random action helps the code 'explore' & find optimum solution