

- *Randomness Injection*
- *Feature-Selection Ensembles*
- *Error-Correcting Output Coding*

Methods for Coordinated Construction of Ensembles

UNIT – V

Unsupervised Learning : Clustering-K-means, K-Modes, K-Prototypes, Gaussian

Mixture Models, Expectation-Maximization.

Reinforcement Learning: Exploration and exploitation trade-offs, non-associative learning, Markov decision processes, Q-learning.

Unsupervised Machine Learning:

Introduction to clustering

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

“Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.”

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format**

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.



Why use Unsupervised Learning?

Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.

- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

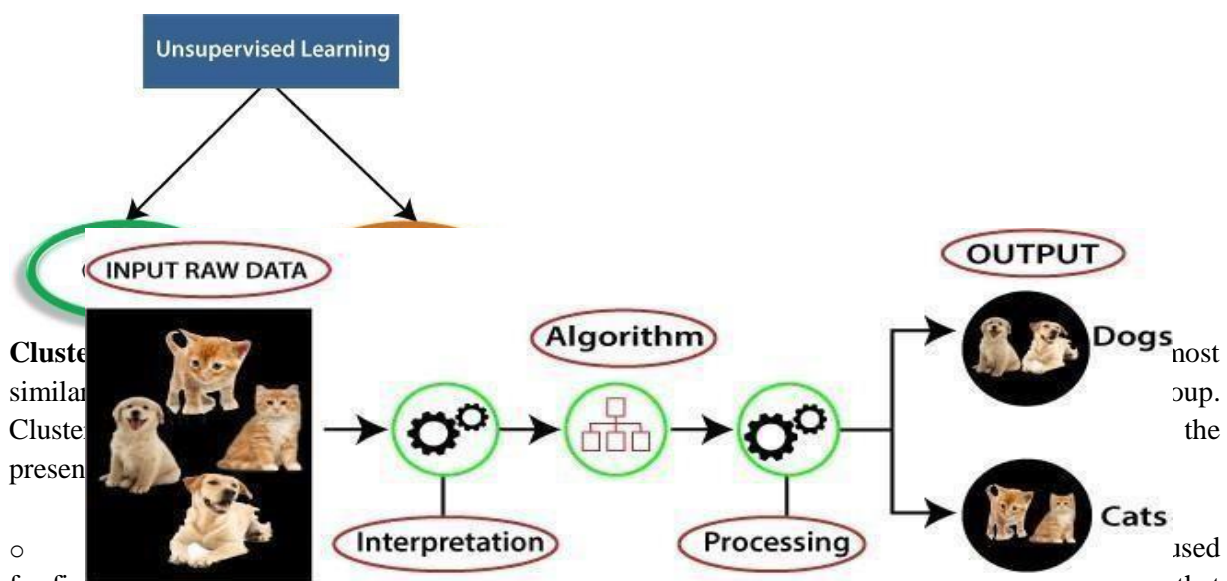
Working of Unsupervised Learning

Work Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

Types of Unsupervised Learning Algorithm:

The unsupervised learning algorithm can be further categorized into two types of problems:



○ for finding the unlabeled data that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

K-means clustering of unsupervised learning can be understood by the below diagram:

- KNN (k-nearest neighbors)
- Hierarchical clustering
- Anomaly detection
- Neural Networks
- Principle Component Analysis
- Independent Component Analysis
- Apriori algorithm
- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages of Unsupervised Learning

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

k-means clustering algorithm

One of the most used clustering algorithm is *k-means*. It allows to group the data according to the existing similarities among them in k clusters, given as input to the algorithm. I'll start with a simple example.

Let's imagine we have 5 objects (say 5 people) and for each of them we know two features (height and weight). We want to group them into $k=2$ clusters.

Our dataset will look like this:

How to apply k-means?

As you probably already know, I'm using Python libraries to analyze my data. The *k-means* algorithm is implemented in the *scikit-learn* package. To use it, you will just need the following line in your script:

What if our data is... non-numerical?

At this point, you will maybe have noticed something. The basic concept of *k-means* stands on mathematical calculations (means, euclidian distances). But what if our data is non-numerical or, in other words, *categorical*? Imagine, for instance, to have the ID code and date of birth of the five people of the previous example, instead of their heights and weights.

We could think of transforming our categorical values in numerical values and eventually apply *k-means*. But beware: *k-means* uses numerical distances, so it could consider close two really distant objects that merely have been assigned two close numbers.

k-modes is an extension of *k-means*. Instead of distances it uses *dissimilarities* (that is, quantification of the total mismatches between two objects: the smaller this number, the more similar the two objects). And instead of means, it uses *modes*. A mode is a vector of elements that minimizes the dissimilarities between the vector itself and each object of the data. We will have as many modes as the number of clusters we required, since they act as centroids.

Reinforcement learning

Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals

Introduction

- Consider building a **learning robot**. The robot, or **agent**, has a set of sensors to observe the state of its environment, and a set of actions it can perform to alter this state.
- Its task is to learn a control strategy, or **policy**, for choosing actions that achieve its goals.
- The goals of the agent can be defined by a **reward function** that assigns a numerical value to each distinct action the agent may take from each distinct state.
- This reward function may be built into the robot, or known only to an external teacher who provides the reward value for each action performed by the robot.
- The **task** of the robot is to perform sequences of actions, observe their consequences, and learn a control policy.
- The control policy is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent.

Example:

- A mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn."
- The robot may have a goal of docking onto its battery charger whenever its battery level is

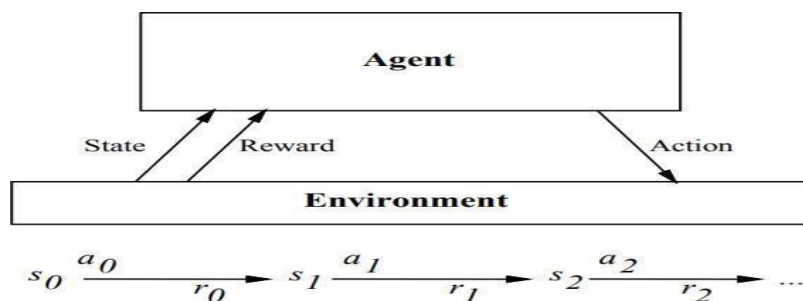
low.

- The goal of docking to the battery charger can be captured by assigning a positive reward (Eg., +100) to state- action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition.

Reinforcement Learning Problem

- An agent interacting with its environment. The agent exists in an environment described by some set of possible states S .
- Agent perform any of a set of possible actions A . Each time it performs an action a , in some state s_t the agent receives a real-valued reward r , that indicates the immediate value of this state-action transition. This produces a sequence of states s_i , actions a_i , and immediate rewards r_i as shown in the figure.

The agent's task is to learn a control policy, $\pi: S \rightarrow A$, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.



Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots, \text{ where } 0 \leq \gamma < 1$$

Reinforcement learning problem characteristics

1. **Delayed reward:** The task of the agent is to learn a target function π that maps from the current state s to the optimal action $a = \pi(s)$. In reinforcement learning, training information is not available in $(s, \pi(s))$. Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions. The agent, therefore, faces the problem of **temporal credit assignment**: determining which of the actions in its sequence are to be credited with producing the eventual rewards.

2. **Exploration:** In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses. This raises the question of which experimentation strategy produces most effective learning. The learner faces a trade-off in choosing whether to favor exploration of unknown states and actions, or exploitation of states and actions that it has already learned will yield high reward.

3. **Partially observable states:** The agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information. In such cases, the agent needs to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to

improve the observability of the environment.

4. **Life-long learning:** Robot requires to learn several related tasks within the same environment, using the same sensors. For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers. This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

Learning Task

Consider Markov decision process (MDP) where the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform

- At each discrete time step t , the agent senses the current state s_t , chooses a current action a_t , and performs it.
- The environment responds by giving the agent a reward $r_t = r(s_t, a_t)$ and by producing the succeeding state $s_{t+1} = \delta(s_t, a_t)$. Here the functions $\delta(s_t, a_t)$ and $r(s_t, a_t)$ depend only on the current state and action, and not on earlier states or actions.

The task of the agent is to learn a policy, $\pi: S \rightarrow A$, for selecting its next action a , based on the current observed state s_t ; that is, $\pi(s_t) = a_t$.

How shall we specify precisely which policy π we would like the agent to learn?

1. One approach is to require the policy that produces the greatest possible **cumulative reward** for the robot over time.

[To state this requirement more precisely, define the cumulative value $V^\pi(s_t)$ achieved by following an arbitrary policy π from an arbitrary

$$\begin{aligned} V^\pi(s_t) &\equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned} \quad \text{equ (1)}$$

initial state s_t as follows:

[Where, the sequence of rewards r_{t+i} is generated by beginning at state s_t and by repeatedly using the policy π to select actions.

[Here $0 \leq \gamma \leq 1$ is a constant that determines the relative value of delayed versus immediate rewards. if we set γ

$= 0$, only the immediate reward is considered. As we set γ closer to 1, future rewards are given greater emphasis relative to the immediate reward.

[The quantity $V^\pi(s_t)$ is called the **discounted cumulative reward** achieved by policy π

from initial state s . It is reasonable to discount future rewards relative to immediate rewards because, in many cases, we prefer to obtain the reward sooner rather than later.

2. Other definitions of total reward is *finite horizon reward*,

$$\sum_{i=0}^h r_{t+i}$$

Considers the undiscounted sum of rewards over a finite number h of steps

3. Another approach is *average reward*

$$\lim_{h \rightarrow \infty} \frac{1}{h} \sum_{i=0}^h r_{t+i}$$

Considers the average reward per time step over the entire lifetime of the agent.

We require that the agent learn a policy π that maximizes $V\pi(s)$ for all states s . such a policy is called an *optimal policy* and denote it by π^*

Refer the value function $V\pi^*(s)$ an optimal policy as $V^*(s)$. $V^*(s)$ gives the maximum discounted cumulative reward that the agent can obtain starting from state s .

Example:

A simple grid-world environment is depicted in the diagram

- [The six grid squares in this diagram represent six possible states, or locations, for the agent.
- [Each arrow in the diagram represents a possible action the agent can take to move from one state to another.
- [The number associated with each arrow represents the immediate reward $r(s, a)$ the agent receives if it executes the corresponding state-action transition
- [The immediate reward in this environment is defined to be zero for all state-action transitions except for those leading into the state labelled G. The state G as the goal state, and the agent can receive reward by entering this state.

Once the states, actions, and immediate rewards are defined, choose a value for the discount factor γ , determine the optimal policy π^* and its value function $V^*(s)$.

Let's choose $\gamma = 0.9$. The diagram at the bottom of the figure shows one optimal

policy for this setting.

Values of $V^*(s)$ and $Q(s, a)$ follow from $r(s, a)$, and the discount factor $\gamma = 0.9$. An optimal policy, corresponding to actions with maximal Q values, is also shown.

The discounted future reward from the bottom centre state is

$$0 + \gamma 100 + \gamma^2 0 + \gamma^3 0 + \dots = 90$$

***Q* LEARNING**

How can an agent learn an optimal policy π^ for an arbitrary environment?*

The training information available to the learner is the sequence of immediate rewards $r(s_i, a_i)$ for $i = 0, 1, 2, \dots$.

Given this kind of training information it is easier to learn a numerical evaluation function defined over states and actions, then implement the optimal policy in terms of this evaluation function.

What evaluation function should the agent attempt to learn?

One obvious choice is V^* . The agent should prefer state s_1 over state s_2 whenever $V^*(s_1) > V^*(s_2)$, because the cumulative future reward will be greater from s_1 .

The optimal action in state s is the action a that maximizes the sum of the immediate reward $r(s, a)$ plus the value V^* of the immediate successor state, discounted by γ .

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} [r(s, a) + \gamma V^*(\delta(s, a))] \quad \text{equ (3)}$$

The Q Function

The value of Evaluation function $Q(s, a)$ is the reward received immediately upon executing action a from state s , plus the value (discounted by γ) of following the optimal policy thereafter

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a)) \quad \text{equ (4)}$$

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q(s, a) \quad \text{equ (5)}$$

Rewrite Equation (3) in terms of $Q(s, a)$ as

Equation (5) makes clear, it need only consider each available action a in its current state s and choose the action that maximizes $Q(s, a)$.

An Algorithm for Learning Q

- Learning the Q function corresponds to learning the **optimal policy**.
- The key problem is finding a reliable way to estimate training values for Q , given only a sequence of immediate rewards r spread out over

$$V^*(s) = \max_{a'} Q(s, a')$$

time. This can be accomplished through *iterative approximation*

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

Rewriting Equation

- Q learning algorithm:

Q learning algorithm

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero.

Observe the current state s

Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$
-

Q learning algorithm assuming deterministic rewards and actions.

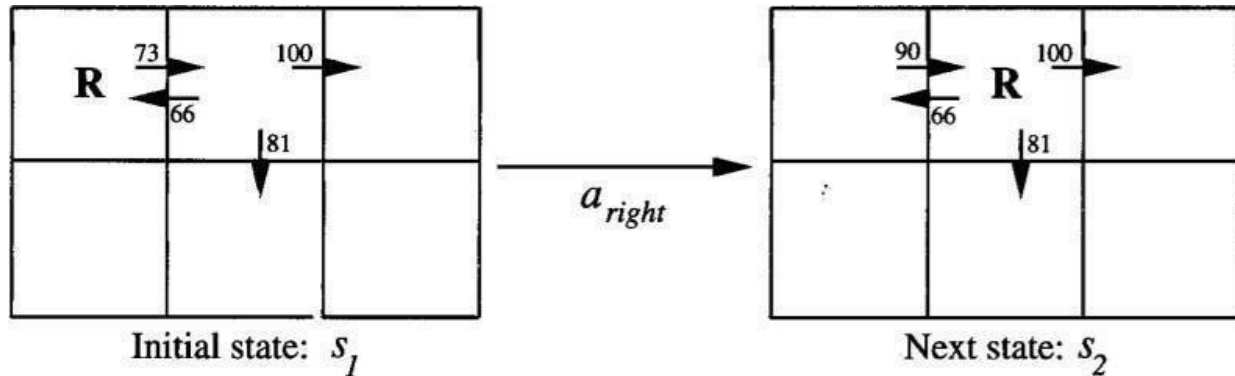
The discount factor γ may be any constant such that $0 \leq \gamma < 1$

- \hat{Q} to refer to the learner's estimate, or hypothesis, of the actual Q function

An Illustrative Example

- To illustrate the operation of the Q learning algorithm, consider a single action taken by an agent, and the corresponding refinement to

Q shown in below figure



The agent moves one cell to the right in its grid world and receives an immediate reward of zero for this transition.

Apply the training rule of Equation

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

to refine its estimate Q for the state-action transition it just executed.

$$\begin{aligned} \hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \max\{66, 81, 100\} \\ &\leftarrow 90 \end{aligned}$$

According to the training rule, the new Q estimate for this transition is the sum of the received reward (zero) and the highest Q value associated with the resulting state (100), discounted by γ (.9).

Convergence

Will the Q Learning Algorithm converge toward a Q equal to the true Q function?

Yes, under certain conditions.

1. Assume the system is a deterministic MDP.
2. Assume the immediate reward values are bounded; that is, there exists some positive constant c such that for all states s and actions a , $|r(s, a)| < c$
3. Assume the agent selects actions in such a fashion that it visits every possible state-action pair infinitely often

Here are four machine learning trends that could become a reality in the near future:

1) Intelligence on the Cloud

Algorithms can help companies unearth insights about their business, but this proposition can be expensive with no guarantees of a bottom-line increase. Companies often deal with having to collect data, hire data scientists and train them to deal with changing databases. Now that more data metrics are becoming available, the cost to store it is dropping thanks to the cloud. There will no longer be the need to manage infrastructure as cloud systems can generate new models as the scale of an operation increases, while also delivering more accurate results. More open-source ML frameworks are coming to the fold, obtaining pre-trained platforms that can tag images, recommend products and perform natural language processing tasks.

2) Quantum Computing Capabilities

Some of the tasks that ML can help companies deal with is the manipulation and classification of large quantities of vectors in high-dimensional spaces. Current algorithms take a large chunk of time to solve these problems, costing companies more to complete their business processes. Quantum computers are slated to become all the rage soon as they can manipulate high-dimensional vectors at a fraction of the time. These will be able to increase the number of vectors and dimensions that are processed when compared to traditional algorithms in a quicker period of time.

3) Improved Personalization

Retailers are already making waves in developing recommendation engines that reach their target audience more accurately. Taking this a step further, ML will be able to improve the personalization techniques of these engines in more precise ways. The technology will offer more specific data that they can then use on ads to improve the shopping experience for consumers.

4) Data on Data

As the amount of data available increases, the cost of storing this data decreases at roughly the same rate. ML has great potential in generating data of the highest quality that will lead to better models, an improved user experience and more data that helps repeat but improve upon this cycle. Companies such as Tesla add a million miles of driving data to enhance its self-driving capabilities every hour. Its Autopilot feature learns from this data and improves the software that propels these self-driving vehicles forward as the company gathers more data on the possible pitfalls of autonomous driving technology.