Machine Learning (21AI63)

Instance Based Learning

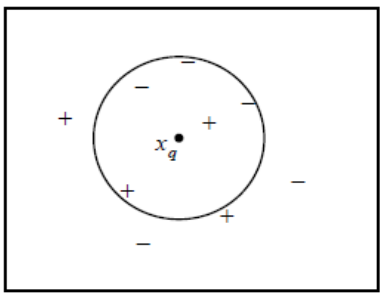
- Instance-Based Learning (IBL) is a type of supervised learning where the system learns from examples and makes predictions based on similarity measures.
- Instance-Based Learning works by storing instances of training data and making predictions for new data points based on similarity measures (e.g., distance metrics like Euclidean distance).
- Unlike traditional machine learning algorithms that involve a distinct training phase, IBL systems do not explicitly build a model. Instead, they store instances of training data and use them directly during prediction.
- IBL is sometimes referred to as "lazy learning" because it defers processing (learning) until a new prediction needs to be made. This means it can adapt to new data on the fly without needing to retrain a model.
- Common examples of IBL algorithms include k-Nearest Neighbours (k-NN), where the prediction is based on the majority class among the k-nearest neighbours in the feature space.
- IBL can be effective when the underlying relationship between features and outputs is complex or not well-defined, as it directly uses instances from the training data without assumptions about the data distribution.

K-Nearest Neighbour (KNN)

- k-Nearest Neighbour (k-NN) is one of the simplest and most widely used instance-based learning algorithms in machine learning.
- Supervised learning algorithm used for classification and regression.
- k-NN makes predictions based on the similarity between a new data point and its nearest neighbours in the training set.

Steps in KNN:

- Determine the number of neighbours k. The value of k is a hyperparameter and can significantly impact the performance of the algorithm.
- Select a distance metric to measure the similarity between instances. Common metrics include Euclidean distance.
- For a given new data point, calculate the distance between this point and all instances in the training set.
- Identify the k training instances that are closest to the new data point based on the chosen distance metric.
- For classification tasks, k-NN predicts the class of the new data point by taking a majority vote among the classes of the k nearest neighbours.
- Figure illustrates the operation of the k-Nearest Neighbour algorithm for the case where the instances are points in a two-dimensional space and where the target function is Boolean valued.



- The positive and negative training examples are shown by "+" and "-" respectively. A query point xq is shown as well.
- The 1-Nearest Neighbor algorithm classifies xq as a positive example in this figure, whereas the 5-Nearest Neighbor algorithm classifies it as a negative example.
- The distance between two instances xi and xj is defined to be d(xi , xj).

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Example for KNN

Training Data:

- Point 1: (1, 2), Class = A
- Point 2: (2, 3), Class = A
- Point 3: (3, 1), Class = B
- Point 4: (6, 5), Class = B

New Point to Classify:

- (4, 2)

Solution:

Let's choose k=3

Euclidean Distance Formula: $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Distance between New Point and Point 1: $\sqrt{(1-4)^2 + (2-2)^2} = \sqrt{9+0} = 3$

Distance between New Point and Point 2: $\sqrt{(2-4)^2 + (3-2)^2} = \sqrt{4+1} = \sqrt{5} \approx 2.24$

Distance between New Point and Point 3: $\sqrt{(3-4)^2 + (1-2)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$

Distance between New Point and Point 4: $\sqrt{(6-4)^2 + (5-2)^2} = \sqrt{4+9} = \sqrt{13} \approx 3.61$

- The three nearest neighbours are:
  - Point 3 (distance = 1.41) – Class B
  - Point 2 (distance = 2.24) – Class A
  - Point 1 (distance = 3) – Class A
- Majority Vote:

  Class A: 2 votes

  Class B: 1 vote

- Prediction: The new point is classified as Class A.

Advantages of k-NN

- Easy to understand and implement.
- Can be used for both classification and regression tasks.

- Since it is a lazy learner, there is no separate training phase, and the model is ready as soon as the data is available.
- Naturally adapts to changes in the data since it uses the actual data points for making predictions.

Disadvantages of k-NN

- k-NN can be computationally expensive, especially with large datasets, as it requires calculating the distance between the new data point and all training instances for each prediction.
- Needs to store all training data, which can be memory-intensive.
- The performance of k-NN can degrade in high-dimensional spaces because the distance metric may become less meaningful.
- The performance of k-NN is sensitive to the choice of k. Too small k can lead to noisy predictions, while too large k can cause the predictions to be biased towards the majority class.

KNN algorithm for discrete-valued target function and real-valued target function

KNN algorithm for discrete-valued target function

- Calculate the distance between the query point and all points in the dataset using a distance metric (e.g., Euclidean distance).
- Identify the k points that are closest to the query point.
- Assign the class label that is most common among the k nearest neighbours.

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

KNN algorithm for real-valued target function

- Calculate the distance between the query point and all points in the dataset using a distance metric (e.g., Euclidean distance).
- Identify the k points that are closest to the query point.
- Compute the average of the target values of the k nearest neighbours to predict the target value for the query point.

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Reinforcement Learning

- Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
- In Reinforcement Learning, the agent learns automatically using feedbacks without any labelled data, unlike supervised learning.
- Since there is no labelled data, so the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as game-playing, robotics, etc.
- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.

Terms used in Reinforcement Learning

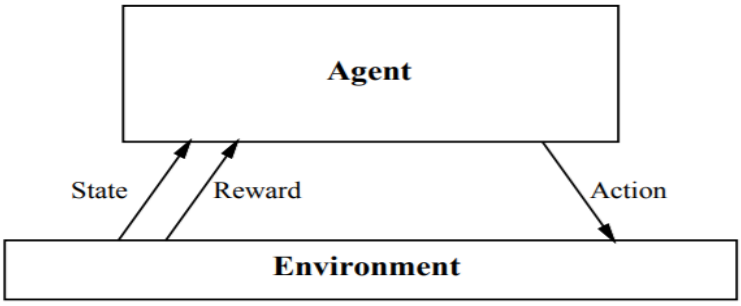
- Agent(): An entity that can perceive/explore the environment and act upon it.
- Environment(): A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- Action(): Actions are the moves taken by an agent within the environment.
- State(): State is a situation returned by the environment after each action taken by the agent.
- Reward(): A feedback returned to the agent from the environment to evaluate the action of the agent.
- Policy(): Policy is a strategy applied by the agent for the next action based on the current state.
- Value(): It is expected long-term retuned with the discount factor and opposite to the short-term reward.
- Q-value(): It is mostly similar to the value, but it takes one additional parameter as a current action.

Key Features of Reinforcement Learning

- In RL, the agent is not instructed about the environment and what actions need to be taken.
- It is based on the hit and trial process.
- The agent takes the next action and changes states according to the feedback of the previous action.
- The agent may get a delayed reward.
- The environment is stochastic, and the agent needs to explore it to reach to get the maximum positive rewards.

Reinforcement Learning Problem

- An agent interacting with its environment. The agent exists in an environment described by some set of possible states S.
- Agent performs any of a set of possible actions A. Each time it performs an action a, in some state st the agent receives a real-valued reward r, that indicates the immediate value of this state-action transition. This produces a sequence of states si, actions ai, and immediate rewards ri as shown in the figure.
- The agent's task is to learn a control policy, π: S → A, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.



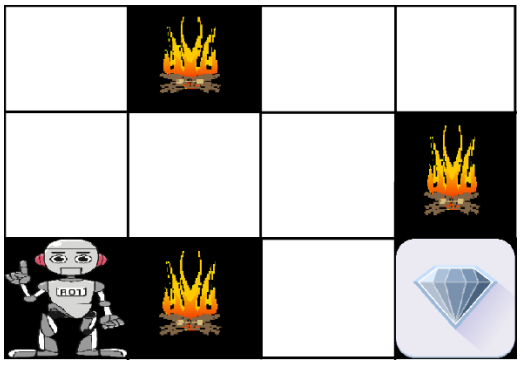
$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$$

Goal: Learn to choose actions that maximize

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

Example:

We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward.



- The above image shows the robot, diamond, and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that are fired.
- The robot learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract the reward of the robot.
- The total reward will be calculated when it reaches the final reward that is the diamond.

Reinforcement learning problem characteristics

- 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 = π (s). In reinforcement learning, training information is not available in (s, π (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.
- 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.
- 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.

Learning tasks and Q-learning

- Q-learning is a popular model-free reinforcement learning algorithm based on the Bellman equation.
- The main objective of Q-learning is to learn the policy which can inform the agent that what actions should be taken for maximizing the reward under what circumstances.
- It is an off-policy RL that attempts to find the best action to take at a current state.
- Q-learning models engage in an iterative process where various components collaborate to train the model. This iterative procedure encompasses the agent exploring the environment and continuously updating the model based on this exploration.
- The goal of the agent in Q-learning is to maximize the value of Q. Here are the two methods to determine the Q-value:
    - Temporal difference. The temporal difference formula calculates the Q-value by incorporating the value of the current state and action by comparing the differences with the previous state and action.
    - Bellman's equation. Mathematician Richard Bellman invented this equation in 1957 as a recursive formula for optimal decision-making. In the q-learning context, Bellman's equation is used to help calculate the value of a given state and assess its relative position. The state with the highest value is considered the optimal state.

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 time. This can be accomplished through ***iterative approximation***

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

Rewriting Equation

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

- **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:

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

- $s \leftarrow s'$

- The Q-table functions as a repository of rewards associated with optimal actions for each state in a given environment. It serves as a guide for the agent, indicating which actions are likely to yield positive outcomes in various scenarios.
- Each row in the Q-table corresponds to a distinct situation the agent might face, while the columns represent the available actions. Through interactions with the environment and the receipt of rewards or penalties, the Q-table is dynamically updated to capture the model's evolving understanding.