Machine Learning

Recently, machine learning has captured the attention in many fields and been utilized in different applications, such as voice, image recognition and stock market prediction. Machine learning can be best defined as a set of methods which are able to extract information or pattern from a given data in order to learn and understand the system under consideration, and if necessary predict its future behavior [Mur12]. However, to ensure reliable functionalities, proper learning algorithms have to be considered based on the problem in hand.

Supervised Learning

Given the pair (x_i, y_i), with $i = 1, \ldots, n$, the set

$$\{(x_i, y_i) \mid i = 1, \ldots, n\}$$

is called the training set of $n \in \mathbb{N}$ number of training examples, where $x_i \in \mathcal{X}$ is known as the input or feature variable, and $y_i \in \mathcal{Y}$ as the output or target variable. Via supervised learning approach, machine learning algorithms utilize such a training set to determine a function or a hypothesis

$$h : \mathcal{X} \longrightarrow \mathcal{Y},$$

such that $h(x_i)$ is a "good" predictor of y_i. Once the learning is over, i.e, the parameters of the function h is determined, it can be utilized to test or identify new samples. Based on the nature of y_i, the problem is

- a regression one, if $\mathcal{Y}$ is continuous, or
- a classification one, if $\mathcal{Y}$ is discrete.

Linear Regression

Given the training set $\{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$, where $\mathbf{x}_i \in \mathbb{R}^p$, and $y_i \in \mathbb{R}$. The linear function

$$\begin{aligned} y_i &= \nu_0 + x_1\nu_1 + \cdots + x_p\nu_p + \epsilon_i \\ &= (1, \mathbf{x}_i^T)\boldsymbol{\nu} + \epsilon_i, \end{aligned}$$

is considered a linear regression model, where $\epsilon_i \in \mathbb{R}$ is a random error. Hence, the learning function is represented by $h_{\boldsymbol{\nu}}(\mathbf{x}) = (1, \mathbf{x}^T)\boldsymbol{\nu}$, where $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_p)^T$ are the regression parameters. Thus, the regression problem can be redefined as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\nu} + \boldsymbol{\epsilon}, \tag{7.1}$$

where

$$\mathbf{X}_{n\times(p+1)} = \begin{pmatrix} 1 & \mathbf{x}_1^T \\ \vdots & \vdots \\ 1 & \mathbf{x}_n^T \end{pmatrix}, \qquad \mathbf{y} = (y_1, \ldots, y_n)^T, \qquad \text{and} \qquad \boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)^T.$$

In (7.1), the best $\boldsymbol{\nu}$, denoted by $\boldsymbol{\nu}^\star$, is required to be obtained by solving the minimization problem

$$\min_{\boldsymbol{\nu}\in\mathbb{R}^{p+1}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\nu}\|, \tag{7.2}$$

The solution can be performed in the following steps.

1. Find $\hat{\mathbf{y}}$ by projecting $\mathbf{y}$ onto $\mathrm{Im}(\mathbf{X})$
   The term $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is an orthogonal projection onto $\mathrm{Im}(\mathbf{X})$, provided the inverse $(\mathbf{X}^T\mathbf{X})^{-1}$ exists. Note that the inverse $(\mathbf{X}^T\mathbf{X})^{-1}$ must exist, however if not, replace $(\mathbf{X}^T\mathbf{X})^{-1}$ by the so-called Moore-Penrose-inverse $(\mathbf{X}^T\mathbf{X})^{+}$.
   Thus,

$$\begin{aligned} \hat{\mathbf{y}} &= \arg\min_{\mathbf{z}\in\mathrm{Im}(\mathbf{X})} \|\mathbf{y} - \mathbf{z}\| \\ &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

2. Find $\boldsymbol{\nu}$ such that $\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\nu}$

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{X}\boldsymbol{\nu} \\ \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} &= \mathbf{X}\boldsymbol{\nu} \end{aligned}$$

   multiplying both sides by $\mathbf{X}^T$ results

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\boldsymbol{\nu}.$$

   Thus,

$$\boldsymbol{\nu}^\star = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

   and,

$$\mathbf{X}\boldsymbol{\nu}^\star = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{y} = \hat{\mathbf{y}}.$$

As an exercise, prove that $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is the orthogonal projection onto $\mathrm{Im}\mathbf{X}$, provided that $(\mathbf{X}^T\mathbf{X})^{-1}$ exists.

Example. A 1-dimensional regression problem: Figure 7.1 shows a 1-dimensional regression problem, which can be formulated as follows.

$$y_i = \nu_0 + \nu_1 x_i + \epsilon_i, \quad \text{for } i = 1, \ldots, n.$$

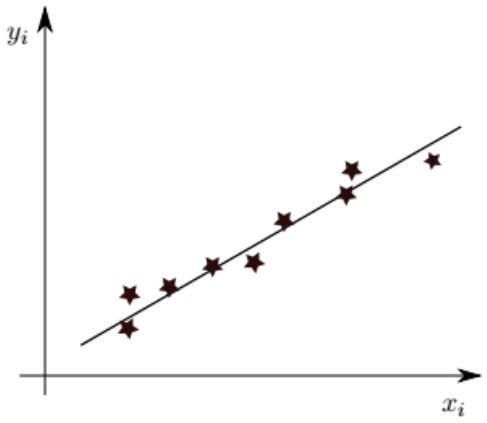

Figure 7.1: A 1-D regression problem.

Find the regression parameters ν_0 and ν_1.

Solution: Given that

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \qquad \text{and,} \qquad \boldsymbol{\nu} = \left(\nu_0, \nu_1\right)^T,$$

the term $\mathbf{X}^T\mathbf{X}$ is obtained as

$$\begin{aligned} \mathbf{X}^T\mathbf{X} &= \begin{pmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_n \end{pmatrix} \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \\ &= \begin{pmatrix} n & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{pmatrix}, \end{aligned}$$

hence

$$\begin{aligned} (\mathbf{X}^T\mathbf{X})^{-1} &= \frac{1}{n\sum_i x_i^2 - (\sum_i x_i)^2} \begin{pmatrix} \sum_i x_i^2 & -\sum_i x_i \\ -\sum_i x_i & n \end{pmatrix} \\ &= \frac{\frac{1}{n^2}}{\frac{1}{n}\sum_i x_i^2 - (\frac{1}{n}\sum_i x_i)^2} \begin{pmatrix} \sum_i x_i^2 & -\sum_i x_i \\ -\sum_i x_i & n \end{pmatrix} \\ &= \frac{1}{n}\frac{1}{\overline{x^2} - \overline{x}^2} \begin{pmatrix} \overline{x^2} & -\overline{x} \\ -\overline{x} & 1 \end{pmatrix}. \end{aligned}$$

Thus,

$$\begin{aligned}
\boldsymbol{\nu}^\star &= \begin{pmatrix} \nu_0^\star \\ \nu_1^\star \end{pmatrix} \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y \\
&= \frac{1}{n}\frac{1}{\overline{x^2} - \overline{x}^2}\begin{pmatrix} \overline{x^2} & -\overline{x} \\ -\overline{x} & 1 \end{pmatrix}\begin{pmatrix} n\overline{y} \\ \sum_i x_i y_i \end{pmatrix} \\
&= \frac{1}{\sigma_x^2}\begin{pmatrix} \overline{x^2} & -\overline{x} \\ -\overline{x} & 1 \end{pmatrix}\begin{pmatrix} \overline{y} \\ pxy \end{pmatrix} \\
&= \frac{1}{\sigma_x^2}\begin{pmatrix} \overline{x^2}\,\overline{y} - \overline{x}\,p\,x\,y \\ -\overline{x}\,\overline{y} + p\,x\,y \end{pmatrix} \\
&= \frac{1}{\sigma_x^2}\begin{pmatrix} \overline{x^2}\,\overline{y} - \overline{x}\,p\,x\,y \\ \sigma_{xy} \end{pmatrix}.
\end{aligned}$$

Hence, the regression parameters for a 1-dimensional problem can be directly calculated as

$$\nu_1^\star = \frac{\sigma_{xy}}{\sigma_x^2},$$

and

$$\nu_0^\star = \overline{y} - \nu_1^\star \overline{x},$$

where,

$$\begin{aligned}
\bar{x} &= \frac{1}{n}\sum_i x_i, \\
\bar{y} &= \frac{1}{n}\sum_i y_i, \\
\sigma_{xy} &= \frac{1}{n}\sum_i x_i y_i - \bar{x}\bar{y}, \\
\sigma_{x^2} &= \frac{1}{n}\sum_i x_i^2 - \bar{x}^2.
\end{aligned}$$

Prove as an exercise, that

$$\begin{aligned}
\nu_0^\star &= \overline{y} - \nu_1^\star \overline{x} \\
&= \frac{1}{\sigma_x^2}(\overline{x^2}\,\overline{y} - p\,x\,y\overline{x}).
\end{aligned}$$

Logistic Regression

The concept of regression can be extended for binary classification problems, i.e., problems where the output variable y can be either 0 or 1. Correspondingly, the hypothesis is give

by

$$h_{\boldsymbol{\nu}}(\mathbf{x}) = g(\boldsymbol{\nu}^T\mathbf{x}) = \frac{1}{1+\mathbb{E}(-\boldsymbol{\nu}^T\mathbf{x})} \in [0,1], \tag{7.3}$$

where $g(z) = \frac{1}{1+\mathbb{E}(-z)}$ is called the logistic or sigmoid function. Figure 7.2 represents the logistic function $g(z)$ for values $z \in [-5,5]$. As an exercise, prove that the statement $g'(z) = g(z)(1-g(z))$ holds.

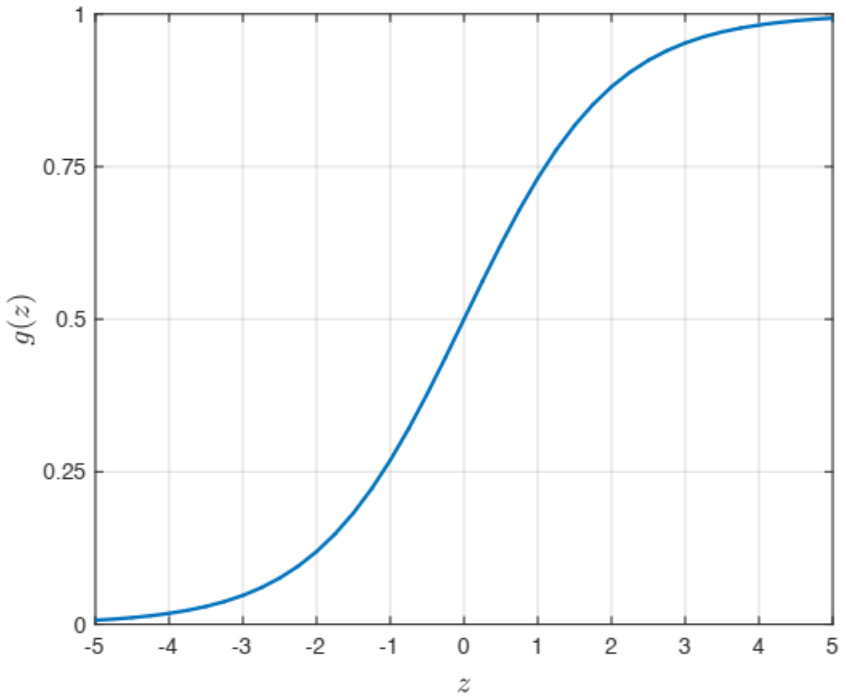

Figure 7.2: The logistic or sigmoid function $g(z)$ for $z \in [-5,5]$.

Given the training set $\{(\mathbf{x}_i, y_i) \mid i = 1,\ldots,n\}$ training samples, where $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{0,1\}$. Similar to the linear regression, we set $x_{i,0} = 1$, subject to

$$\boldsymbol{\nu}^T\mathbf{x} = \nu_0 + \sum_{j=1}^{p} \nu_j x_j, \quad \text{where } \mathbf{x} = (1, x_1, \ldots, x_p).$$

Probabilistic Interpretation of Logistic Regression

The posterior probability of class $y = 1$ is interpreted via the logistic function, such that

$$\begin{aligned} \mathrm{P}(y=1|\,\mathbf{x},\boldsymbol{\nu}) &= h_{\boldsymbol{\nu}}(\mathbf{x}) \\ \mathrm{P}(y=0|\,\mathbf{x},\boldsymbol{\nu}) &= 1 - h_{\boldsymbol{\nu}}(\mathbf{x}). \end{aligned}$$

In other words, the posterior probability can be written as

$$\mathrm{P}(y=1|\,\mathbf{x},\boldsymbol{\nu}) = (h_{\boldsymbol{\nu}}(\mathbf{x}))^y\,(1-h_{\boldsymbol{\nu}}(\mathbf{x}))^{1-y}, \qquad y \in \{0,1\}.$$

Assuming $n \in \mathbb{N}$ independent training samples, such that

$$\mathbf{X} = \begin{pmatrix} 1 & \mathbf{x}_1^T \\ \vdots & \vdots \\ 1 & \mathbf{x}_n^T \end{pmatrix} \in \mathbb{R}^{n\times(p+1)}$$

or equivalently $\mathbf{X} = (x_{i,j})_{\substack{1\le i\le n \\ 0\le j\le p}}$. The corresponding likelihood function is given as

$$\begin{aligned} L(\boldsymbol{\nu}) = \mathrm{P}(\mathbf{y}|\,\mathbf{X},\boldsymbol{\nu}) &= \prod_{i=1}^{n} \mathrm{P}(y_i|\,\mathbf{x}_i,\boldsymbol{\nu}) \\ &= \prod_{i=1}^{n} (h_{\boldsymbol{\nu}}(\mathbf{x}_i))^{y_i}(1-h_{\boldsymbol{\nu}}(\mathbf{x}_i))^{1-y_i}, \end{aligned}$$

and the log-likelihood as

$$\begin{aligned} \ell(\boldsymbol{\nu}) &= \log L(\boldsymbol{\nu}) \\ &= \sum_{i=1}^{n} y_i \log h_{\boldsymbol{\nu}}(\mathbf{x}_i) + (1-y_i)\log(1-h_{\boldsymbol{\nu}}(\mathbf{x}_i)). \end{aligned} \tag{7.4}$$

In order to estimate of the logistic regression parameters $\boldsymbol{\nu}$, the following objective function is considered

$$\max_{\boldsymbol{\nu}\in\mathbb{R}^{p+1}} \ell(\boldsymbol{\nu}).$$

For this matter, the gradient ascent optimization has been considered. The $(k+1)^{\text{th}}$ update of the logistic parameters, denoted by $\boldsymbol{\nu}^{(k+1)}$, are given by

$$\boldsymbol{\nu}^{(k+1)} = \boldsymbol{\nu}^{(k)} + \alpha\, \nabla_{\boldsymbol{\nu}}\, \ell(\boldsymbol{\nu}^{(k)}),$$

where α is the learning parameters. By setting $(\mathbf{x}_i, y_i) = (\mathbf{x}, y)$, with

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{pmatrix} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{pmatrix},$$

and for each addent in (7.4), $\nabla_{\boldsymbol{\nu}}$ is calculated as

$$\begin{aligned} \frac{\partial}{\partial \nu_j}\left[y\log h_{\boldsymbol{\nu}}(\mathbf{x}) + (1-y)\log(1-h_{\boldsymbol{\nu}}(\mathbf{x}))\right] &= A(\boldsymbol{\nu},\mathbf{x})\frac{\partial}{\partial \nu_j} g(\boldsymbol{\nu}^T\mathbf{x}) \\ &= A(\boldsymbol{\nu},\mathbf{x}) g(\boldsymbol{\nu}^T\mathbf{x})(1-g(\boldsymbol{\nu}^T\mathbf{x}))\frac{\partial}{\partial \nu_j}\boldsymbol{\nu}^T\mathbf{x} \\ &= \left(y(1-g(\boldsymbol{\nu}^T\mathbf{x})) - (1-y)g(\boldsymbol{\nu}^T\mathbf{x})\right)\mathbf{x}_j \\ &= (y - h_{\boldsymbol{\nu}}(\mathbf{x}))\mathbf{x}_j, \end{aligned}$$

where

$$A(\boldsymbol{\nu}, \mathbf{x}) = y\frac{1}{g(\boldsymbol{\nu}^T\mathbf{x})} - (1-y)\frac{1}{1-g(\boldsymbol{\nu}^T\mathbf{x})},$$

and

$$\mathbf{x}_j = \begin{pmatrix} 1 \\ x_{1,j} \\ \vdots \\ x_{p,j} \end{pmatrix}.$$

Hence,

$$\frac{\partial}{\partial \nu_j}\ell(\boldsymbol{\nu}) = \sum_{i=1}^{n}(y_i - h_{\boldsymbol{\nu}}(\mathbf{x}_i))x_{i,j},$$

with the update rule

$$\nu_j^{(k+1)} := \nu_j^{(k)} + \alpha\sum_{i=1}^{n}(y_i - h_{\boldsymbol{\nu}}(\mathbf{x}_i))x_{i,j}.$$

Alternative approach: using Newton's method. The parameter updates are hence given by

$$\boldsymbol{\nu}^{(k+1)} := \boldsymbol{\nu}^{(k)} - H^{-1}\nabla_{\boldsymbol{\nu}}\ell(\boldsymbol{\nu}),$$

where H is the Hessian matrix.
Furthermore, as (7.3) is utilized to represent the posterior probability, such that

$$\mathrm{P}(y|\,\mathbf{x}, \boldsymbol{\nu}) = (h_{\boldsymbol{\nu}}(\mathbf{x}))^y\,(1-h_{\boldsymbol{\nu}}(\mathbf{x}))^{1-y}, \qquad y \in \{0,1\},$$

hence it can be interpreted via the Bernoulli distribution, as

$$f(k) = p^k(1-p)^{1-k},$$

where $p \in (0,1), k \in \{0,1\}$, and $f(k)$ is given by

$$\begin{aligned} f(k) &= \mathbb{E}(k\log p + (1-k)\log(1-p)) \\ &= \mathbb{E}(k\log\frac{p}{1-p} + \log(1-p)) \\ &= \mathbb{E}(k\log\frac{p}{1-p})\mathbb{E}(1-p), \end{aligned}$$

with $\log\frac{p}{1-p} = q \Leftrightarrow p = \frac{1}{1+\mathbb{E}(-p)}$.

The Perceptron Learning Algorithms

Another approach for binary classification method is via the Perceptron learning algorithm. The function $h_{\boldsymbol{\nu}}(\mathbf{x}) = g(\boldsymbol{\nu}^T\mathbf{x})$ forces the output variable y to be either 0 or 1, i.e, $y = \{0, 1\}$, as

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0, \end{cases}$$

unlike the logistic regression, where $g(z) \in [0, 1]$. In analogy to the logistic regression, the update rule is given by

$$\nu_j^{(k+1)} := \nu_j^{(k)} + \sum_{i=1}^{n} \alpha(y_i - h_{\boldsymbol{\nu}}(\mathbf{x}_i))x_{i,j},$$

where $i = 1, \ldots, n$ and $j = 1, \ldots, p + 1$. The perceptron algorithm is considered to be a rough model of how neurons in the human brain works. Thus, it is one of the initial artificial neural networks with a probabilistic interpretation, without the need of MLE.

Reinforcement Learning

So far, the presented machine learning algorithms employ supervised learning methods, i.e., they rely on having a training set of a certain number of examples in order to estimate their parameters. This section presents the so-called *reinforcement learning* approach. Unlike the supervised learning, it does not need a training set to earn the knowledge of the system under consideration. This approach has to find a suitable action based on the current situation or environment to minimize a certain reward function [Bis06]. In other words, the learning is performed via a trial and error process, where its reliability and accuracy are indicated via the reward function. Applications on the reinforcement learning can be: robot leg coordination, autonomous flying, cell phone routing, factory control, etc,. As no training set is utilized, a reward function is instead considered to give an indication about the learning method itself: good or bad. For an analytical description, Markov decision process (MDP) is to be presented.

Markov Decision Process (MDP)

MDP can be represented via the tuple $(S, A, \{P_{sa}\}, \gamma, \mathbf{R})$, where

- S is a set of states.
- A is a set of actions.
- $P_{s,a}$ transition probabilities indicating the probability the system is presented in state $s \in S$ and taking the action $a \in A$. $P_{s,a}$ is a probability distribution over the state space.

- $\gamma \in [0,1]$ is a discount factor.
- R is a reward function, denoted by $R : S \times A \longrightarrow \mathbb{R}$, or simply by $R : S \longrightarrow \mathbb{R}$.

The dynamics of the process can be represented as

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

where $s_0, s_1, s_2, s_3, \cdots \in S$, and $a_0, a_1, a_2, a_3 \cdots \in A$. The total payoff is given by

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \gamma^3 R(s_3, a_3) + \dots,$$

or simply by

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3) + \dots,$$

if the dependency is on the states only. The goal of the reinforcement learning can be formulated as

$$\max \mathbb{E}(\mathbf{R}(s_0) + \gamma \mathbf{R}(s_1) + \gamma^2 \mathbf{R}(s_2) + \gamma^3 R(s_3) + \dots),$$

over all actions in A. For this matter, a policy function π is defined as

$$\pi : S \longrightarrow A : s \longrightarrow \pi(s),$$

where the state $s \in S$ takes an action $a = \pi(s)$. Furthermore, the value function for the policy π is defined by

$$V^{\pi}(s) = \mathbb{E}(R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3) + \cdots \mid s_0 = s, \pi),$$

which represent the expected total payoff upon starting in state s, and applying the policy π. Note that for any policy π, the Bellman equations hold, i.e.,

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s,\pi(s)}(s') V^{\pi}(s'). \tag{7.5}$$

The right-hand side in (7.5) represents an immediate reward $R(s)$ plus the expected sum of future discounted rewards. As an exercise, prove that (7.5) holds.
Furthermore, solving for V^{π} for fixed policy π means to solve $\mid S \mid$ number of linear equations with $\mid S \mid$ variables if $\mid S \mid < \infty$ holds where $S = \{1, \dots, m\}, m \in \mathbb{N}$.
Let $\mathbf{R}$, $\mathbf{\Psi}^{\pi}$ and $\mathbf{V}^{\pi}$ be variables defined as

$$\mathbf{R} = (R(1), \dots, R(m))^T \in \mathbb{R}^m,$$

$$\mathbf{\Psi}^{\pi} = \begin{pmatrix} P_{1,\pi(1)} \\ \vdots \\ P_{m,\pi(m)} \end{pmatrix} \in \mathbb{R}^{m \times m},$$

and

$$\mathbf{V}^{\pi} = (V^{\pi}(1), \dots, V^{\pi}(m))^T \in \mathbb{R}^m,$$

thus (7.5) can be represented in matrix form as

$$\mathbf{V}^{\pi} = \mathbf{R} + \gamma \mathbf{\Psi}^{\pi} \mathbf{V}^{\pi} \Longleftrightarrow \mathbf{V}^{\pi} = (\mathbf{I}_m - \gamma \mathbf{\Psi}^{\pi})^{-1} \mathbf{R}, \tag{7.6}$$

provided the inverse exists. In (7.6), $\mathbf{I}_m$ represents an $m \times m$ identity matrix. Thus, the optimal value function is defined as

$$\begin{aligned} V^{\star}(s) &= \max_{\pi} V^{\pi}(s) \\ &= \max_{\pi} \mathbb{E}(R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \mid s_0 = s, \pi), \end{aligned}$$

which indicates that the maximum expected payoff over all policies π, when starting at state $s \in S$. In analogous to (7.5), it holds that

$$V^{\star}(s) = R(s) + \max_{a \in A} \sum_{s' \in S} P_{s,a}(s') V^{\star}(s').$$

Futhermore, we also define

$$\pi^{\star}(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{s,a}(s') V^{\star}(s'). \tag{7.7}$$

A partial ordering over policies is defined as

$$\pi \geq \pi' \text{ if } V^{\pi}(s) \geq V^{\pi'}(s), \quad \text{for all } s \in S.$$

Theorem 7.1. *a) There exists an optimal policy $\pi^{\star}$ with*

$$\pi^{\star} \geq \pi, \quad \textit{for all policies } \pi, \textit{such that } \pi^{\star} \textit{ is given by (7.7)}$$

b) All optimal policies achieve the same value,

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

In other words, both *a)* and *b)* state that an optimal policy π^{*} is independent of the initial state $s \in S$.
At this point, it is still needed to compute the optimal policy $\pi^{\star}$. For this matter, there are mainly two algorithms: the value iteration, and the policy iteration algorithms.

Computing the optimal policy

The policy iteration algorithm is utilized to find the optimal $\pi^{\star}$. It is indicated in Algorithm 5.
Based on Algorithm 5, it can be note that,
a) for fixed policy π, $\mathbf{V}^{\pi}$ can be computed from (7.5),
b) $V(s')$ is obtained with respect to the actual policy π. Update of π is greedy with respect to $\mathbf{V}$.
After a finite number of iterations, $\mathbf{V}$ converges to $\mathbf{V}^{\star}$, and π to $\pi^{\star}$.

Algorithm 5 Policy iteration algorithm

1: **procedure** INITIALIZE π RANDOMLY
2: **repeat**
3: a) $\mathbf{V} := \mathbf{V}^{\pi}$
4: b) For each $s \in S$, let $\pi(s) := \arg\max_{a \in A} \sum_{s' \in S} P_{s,a}(s')V(s')$
5: **until** Convergence

Model learning for MDPs

In practice, the transition probability $P_{s,a}$ and sometimes the reward function R are unknown. Therefore, it is required to estimate $P_{s,a}(s'), s' \in S$, and $R(s')$ from a given (observed) data, which are obtained by carrying out trials or experiments in the form of

$$s_0^{(\ell)} \xrightarrow{a_0^{(\ell)}} s_1^{(\ell)} \xrightarrow{a_1^{(\ell)}} s_2^{(\ell)} \xrightarrow{a_2^{(\ell)}} s_3^{(\ell)} \ldots,$$

for $\ell = 1, 2, \ldots, \in \mathbb{N}$ number of trails. Thus, the estimate of $P_{s,a}$, denoted by $P^{\star}s, a$ is calculated as

$$P^*_{s,a} = \frac{\text{times took action } a \text{ in state } s \text{ and got to } s'}{\text{times tool action } a \text{ in state } s}.$$

In case of $\frac{0}{0}$, $P^{\star}s, a(s')$ is chosen to be uniformly distributed, such that $P^{\star}_{s,a}(s') = \frac{1}{|S|}$. Similarity, $R(s)$ can be estimated from observed data. Possible algorithm for learning a MDP with unknown $P_{s,a}$ is shown in Algorithm 6.

Algorithm 6 Possible algorithm for learning a MDP with unknown $P_{s,a}$

1: **procedure** INITIALIZE π RANDOMLY
2: **repeat**
3: a) Execute π in the MDP for some number of trials ℓ
4: b) Update the estimates $P^*_{s,a}$(and potentially R)
5: c) Update π using Algorithm 5
6: **until** Convergence