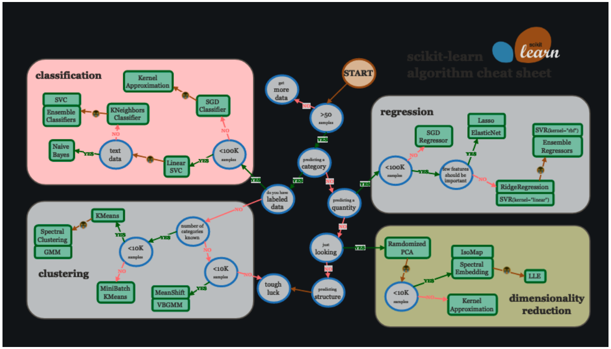
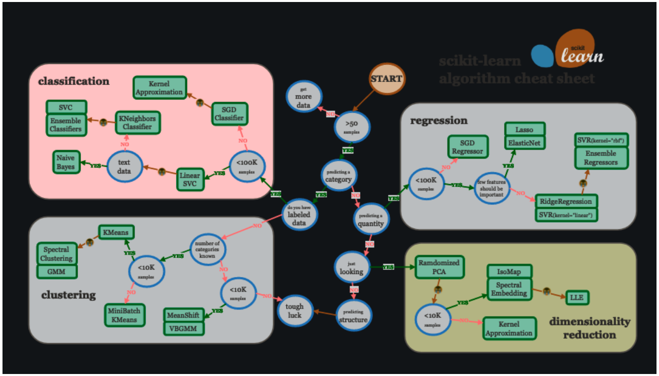
Fondamentaux théoriques du machine learning



Overview of lecture 8

ML map



https://scikit-learn.org/stable/tutorial/machine_learning_map/

No free lunch theorems

$\mathcal{A}$: learning rule. Takes the dataset D_n as input and outputs an estimator f_n (for instance based on empirical risk minimization, local averaging, etc).
There are several no free lunch theorems.

No free lunch theorems

Theorem

No free lunch - fixed n

We consider a binary classification task with "0-1"-loss, and $\mathcal{X}$ infinite.

We note $\mathcal{P}$ the set of all probability distributions on $\mathcal{X} \times \{0, 1\}$.

For any $n > 0$ and any learning rule $\mathcal{A}$

$$\sup_{dp \in \mathcal{P}} E\Big[R_{dp}\big(\mathcal{A}(D_n(dp))\big)\Big] - R^*_{dp} \geq \frac{1}{2} \tag{1}$$

We write $D_n(dp)$ in order to emphasize that the dataset is sampled randomly from the distribution dp.

No free lunch

- For any learning rule, there exists a distribution for which this learning rule performs badly.
- No method is universal and can have a good convergence rate on all problems.

However, considering **all** problems is probably not relevant for machine learning.

Adaptivity

If the learning rule improves (faster convergence rate) when we add a property on the problem (for instance, regularity of the target function), we say that we have adaptivity to this property.
For instance : gradient descent is adaptive to the strong convexity of the target function, since with a proper choice of the learning rate γ, the convergence rate is exponential, with a rate that involves the strong convexity constant μ.
There are several forms of adaptivity.

Most general case

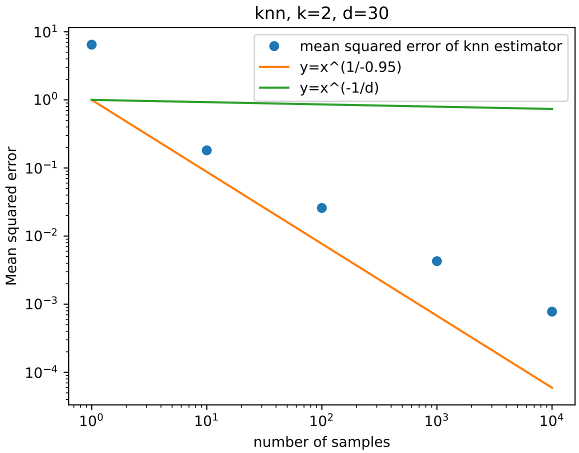
The target is just Lipshitz-continuous, no extra-hypothesis. In this case the optimal rate is of the form $\mathcal{O}(n^{-\frac{1}{d}})$ (curse of dimensionality) for all learning rules.

Adaptivity to the input space

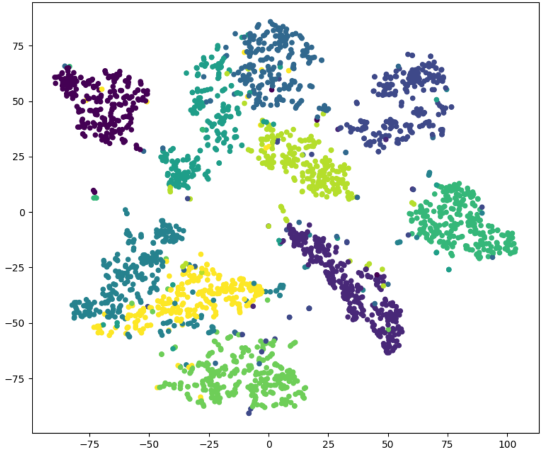
If the input data lie on a submanifold (e.g. a subspace) of $\mathbb{R}^d$ of lower dimension than d, most methods adapt to this property.

Adaptivity to a lower dimensional support

We saw an example of this behavior during a practical seccion.



Lower dimensional manifolds



Adaptivity to the regularity of the target function

If the target is smoother (meaning that all derivatives up to order m are bounded), kernel methods (here, positive-definite kernels) and neural network adapt, if well optimized and regularized.
The convergence rate can become $\mathcal{O}(n^{-\frac{m}{d}})$.

Adaptivity to latent variables

If the target function depends only on a k dimensional linear projection of the data, neural networks adapt, if well optimized. The rate can become $O(n^{-\frac{m}{k}})$.
`https://francisbach.com/quest-for-adaptivity/`

Sutton textbook

http://incompleteideas.net/book/the-book-2nd.html

- RL has many applications and is quite a hot topic.
- **Deep Reinforcement Learning** has received a lot of attention for more than 10 years now.

- Atari games

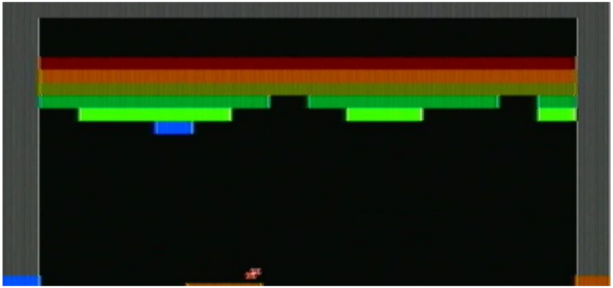


Figure – [Mnih et al., 2013]

- AlphaGo

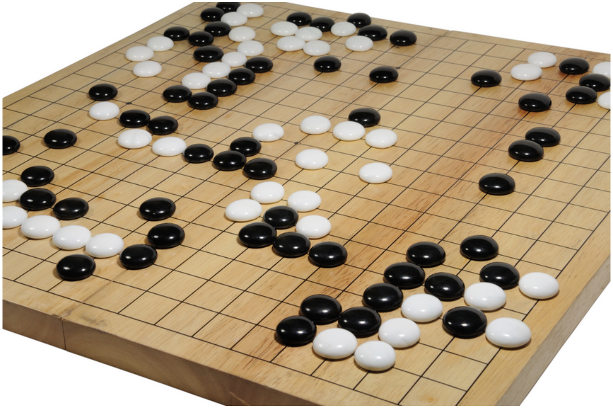


Figure – Go game, beaten by AlphaGo in 2017 [Silver et al., 2016]

- Reinforcement Learning is also being used in the community of **Computationnal neuroscience.**

Supervised learning and Correction

- In **supervised learning**, the supervisor indicates the **expected answer** the model should answer.
- The feedback does not depend on the action performed by the model (for instance the prediction from the model)
- We say that the model receives an **instructive feedback**.
- The model must then **correct its model** based on this answer.

Cost sensitive learning

- In **Cost sensitive learning**, the situation is different.
- The agent receives an **evaluative feedback**. The feedack depends on the action performed by the agent.

Cost sensitive learning

- In **Cost sensitive learning**, the situation is different.
- The agent receives an **evaluative feedback**. The feedack depends on the action performed by the agent.
- **Examples :**
  - AI playing a game and receiving "victory" or "defeat" as a feedback.
  - Child playing with toys.

Reinforcement learning

- **Reinforcement learning** is a particular case of cost-sensitive learning.
- In reinforcement learning, the feedback is a **real number**.

Reinforcement learning

- **Reinforcement learning** is a particular case of cost-sensitive learning.
- In reinforcement learning, the feedback is a **real number**.
- **Example :** amount of coins won after a poker turn.

Reinforcement learning

- First, the agent does not know if a reward is good or bad *per se*.
- A reward of -10 can be good or bad depending on the other rewards that are possible to obtain !

Reinforcement learning

- First, the agent does not know if a reward is good or bad per se.
- A reward of -10 good be good or bad depending on the other rewards that are possible to obtain.
- Most of the time, the objective of the agent will be to optimize the **agregation of rewards**.

Reinforcement learning

- The agent lives in a world E, and can be in several states s. The agent performs **actions** a and receives rewards r.

Reinforcement learning

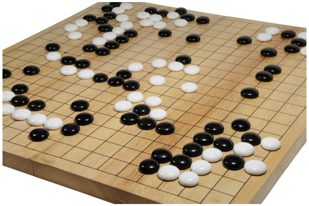
- The agent lives in a world E, and can be in several states s. The agent performs **actions** a and receives rewards r.
- **Examples :**
  - world $= \mathbb{R}^2$
  - state $=$ position
  - actions $=$ moving somewhere
  - reward $=$ amount of food found

Formalization

- There are many aspects of the problem that we need to formalize. Several formalizations are possible depending on the situation.
- We will consider **discrete spaces** :
  - the time will be discrete
  - the number of possible states will be **finite**
  - the number of possible actions will be **finite**
- Continuous spaces are also available for RL. In those cases the objects are slightly different, and the optimization procedures also differ. For an introductory course, discrete spaces are more suitable.

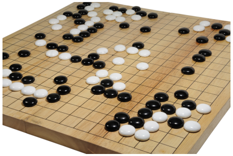
Question

- We will consider **discrete spaces** :
  - the time will be discrete
  - the number of possible states will be **finite**
  - the number of possible actions will be **finite**
- Are these hypotheses valid in the case of AlphaGo ?



Question

- We will consider **discrete spaces** :
  - the time will be discrete
  - the number of possible states will be **finite**
  - the number of possible actions will be **finite**
- Are these hypothesis valid in the case of AlphaGo ?

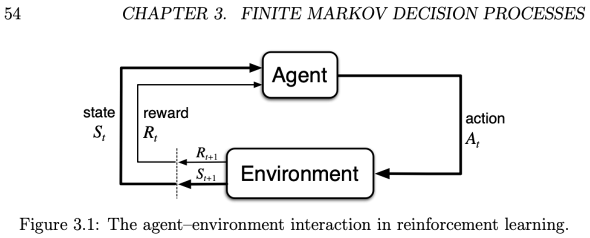


- Yes ! This shows that discrete spaces can still describe very complex problems.

Formalization

- we will write :
  - S_t : state at time t
  - R_t : reward received at time t
  - A_t : action performed at time t
- the actions are chosen according to a **policy** π

Action - reward loop


54 CHAPTER 3. FINITE MARKOV DECISION PROCESSES

Agent
state
S_t
reward
R_t
action
A_t
R_{t+1}
S_{t+1}
Environment

Figure 3.1: The agent–environment interaction in reinforcement learning.

Figure – Image taken from the Sutton book, page 68.

`http://incompleteideas.net/book/the-book-2nd.html`

Policies

- The policy π is a function of the current state.
- It can be **deterministic :** the action chosen is chosen with probability 1.

Policies

- The policy π is a function of the current state.
- It can be **deterministic :** the action chosen is chosen with probability 1.
- Or **stochastic :** the action performed in a given state is drawn from a **distribution**.

Two levels of randomness

- The policy can be deterministic or stochastic.
- But the result of an action could also be stochastic ! This is called a **stochastic transition function**.

Two levels of randomness

- The policy can be deterministic or stochastic.
- But the result of an action could also be stochastic ! This is called a **stochastic transition function**.

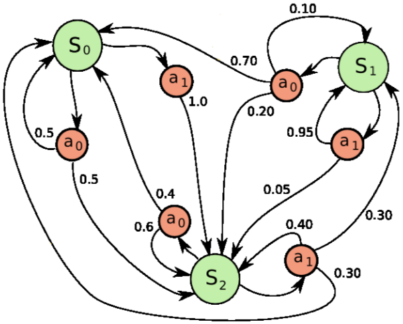


Figure – A stochastic policy with a stochastic transition function.

Exercice 1 :

- What is the probability of staying in state S_0 when performing an action from S_0 ? and from S_1 and S_2 ?

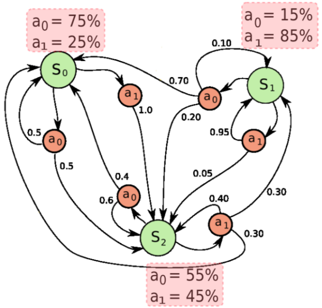


Figure – A stochastic policy with a stochastic transition function.

Agregation of rewards

- The agent want to optimize the **agregation of the rewards.**
- There are several ways to agregate the rewards.

Returns

We introduce the **return** G_t.

- Episodic case (finite number N of steps) :

$$G_t = R_{t+1} + R_{t+2} + \dots + R_{t+N} \tag{2}$$

- Continuing tasks :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{3}$$

$\gamma \in [0, 1[$ is the **discount factor**

Value function

Given a fixed policy π, the value function $v_\pi(s)$ quantifies how good a state s is.

$$v_\pi(s) = E[G_t | S_t = s] \tag{4}$$

(the expectation is taken over the next actions, following the policy π).

The Bellman equation

Given a fixed policy π, and a state s, we can write a recursive relationship between $v_\pi(s)$ and the values v_π of the next possible successor states (see the Sutton book for the general form of the equation 3.12 page 85).

More considerations

- The Markov hypothesis
- Exploitation exploration compromise

ϵ-greedy policy

A way to tackle the exploitation-exploration compromise.

- with probability $1 - \epsilon$: go to the best known reward (exploitation).
- with probability ϵ : perform a random action (exploration).

Art

"RL is a science, but dealing with the exploration-exploitation compromise is an art" (Sutton)

World

We will illustrate reinforcement learning with a specific case (deterministic transition function in a "small" world)

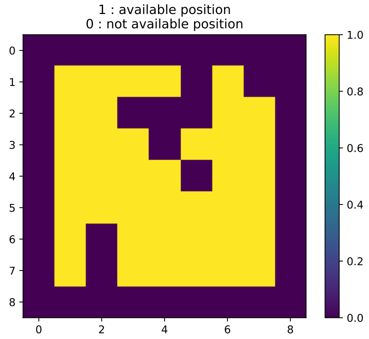


Figure – 2 dimensional world.

Reward

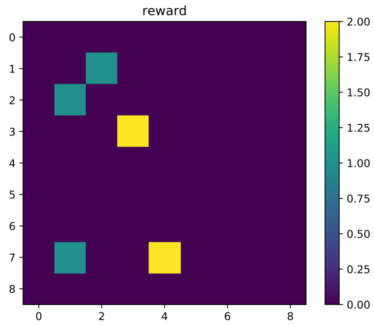


Figure – Reward function.

2D world

- Our agent can move in the 4 directions, one step at a time.

Optimal value functions

We look for the value of the **optimal policy** π^*, defined by the fact that it has the best value function among all policies.

$$
\begin{aligned}
v_*(s) &= \max_{a \in \text{available actions}} E[G_t | S_t = s, A_t = a] \\
&= \max_{a \in \text{available actions}} E[\sum_{k \geq 0} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \\
&= \max_{a \in \text{available actions}} E[R_{t+1} + \gamma \sum_{k \geq 1} \gamma^{k-1} R_{t+k+1} | S_t = s, A_t = a] \\
&= \max_{a \in \text{available actions}} E[R_{t+1} + \gamma \sum_{k \geq 0} \gamma^{k} R_{t+k+2} | S_t = s, A_t = a] \\
&= \max_{a \in \text{available actions}} E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]
\end{aligned} \tag{5}
$$

Bellman optimality equation

In the case of our simple 2D deterministic world, the Bellman optimality equation 5 takes a simpler form !

$$v_*(s) = \max_{a \in \text{available actions}} R_{t+1} + \gamma v_*(S_{t+1}) \tag{6}$$

The expected values are replaced by deterministic values.

Value Iteration

- Value iteration belongs to dynamic programming methods. They are a specific case of RL where a perfect model of the environment is assumed.
- In value iteration, equation 5 is used as an update rule at each time step.

Value Iteration

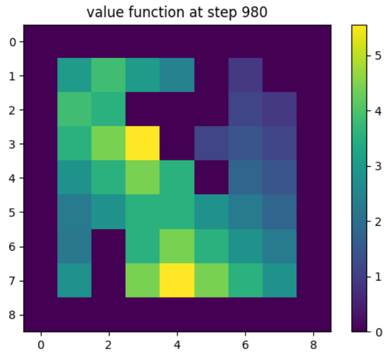
- First, the initial Value function for all the states is 0.
- Then we propagate the information about the rewards between the states, in order to **update the value function** (sweep)

$$\forall s \in V(s) \leftarrow \max_a \left(R_{t+1} + \gamma V(s_{t+1}) | S_t = s, A_t = a\right) \qquad (7)$$

- In parallel, we explore the world to learn about the distribution of rewards.

Value iteration

- After learning, we will obtain a value function



Optimal policy

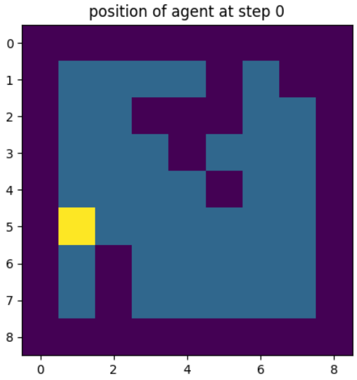


Figure – After learning hte optimal policy, the agent can go to the reward.

Optimal policy

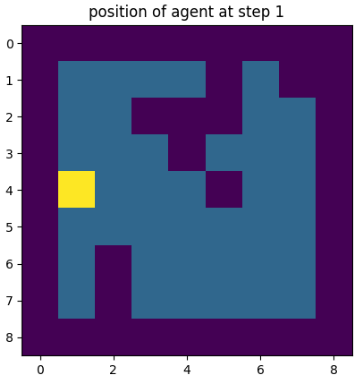


Figure – After learning, the agent can go to the reward.

Optimal policy

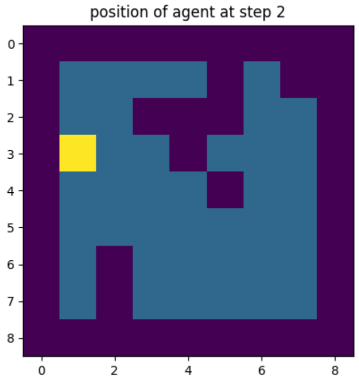


Figure – After learning, the agent can go to the reward.

Optimal policy

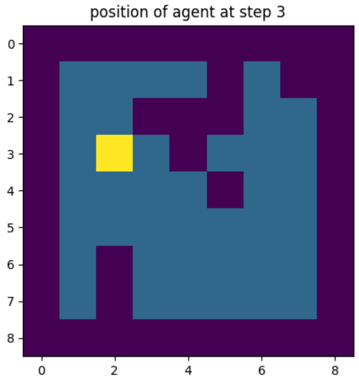


Figure – After learning, the agent can go to the reward.

Optimal policy

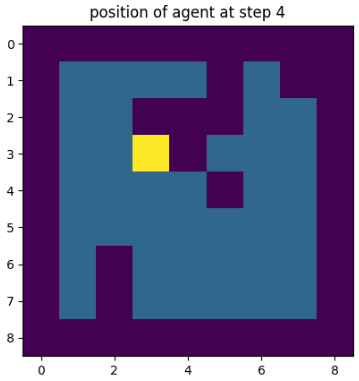


Figure – After learning, the agent can go to the reward.

Optimal policy

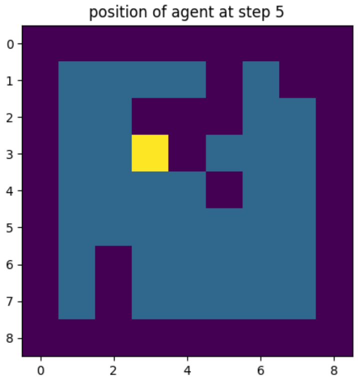


Figure – After learning, the agent can go to the reward.

Multiple paradigms

- Reinforcement learning has many variants.
- In the ones we studied, a model of the consequence of our actions was known.
- This is not always the case.

Temporal difference learning

- In temporal difference learning, the agent does not know a **model** of its world.
- But it can still learn the value function with the **TD (temporal difference) updates**

Temporal difference learning

- In temporal difference learning, the agent does not know a **model** of its world.
- But it can still learn the value function with the **TD (temporal difference) updates**

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \qquad (8)$$

Monte Carlo methods

Monte Carlo methods can be used in Reinforcement Learning to estimate some expected values (such as the expected reward in a given state).

Actor critic methods

- Sometimes you can use **two** policies
    - the **behavior policy** provides actions and guarantees exploration
    - the **target policy** is the optimal policy learned in parallel by the agent, that would be used in exploitation mode.

Tabular case and continous case

- We studied **finite** (and thus discrete situations).
- However, RL can also be applied to continuous state / discrete action spaces (DQN).

Tabular case and continous case

- We studied **finite** (and thus discrete situations).
- However, RL can also be applied to continuous state / discrete action spaces (DQN)
- And even to continous state / continous action spaces (DDPG) [Bengio, 2009] .

Introduction to Kernel methods

Replace inputs $x \in \mathcal{X}$ by a function $\phi(x) \in \mathcal{H}$, with $\mathcal{H}$ a $\mathbb{R}$-Hilbert space. We then perform linear predictions on $\phi(x)$. This means that estimators have the form :

$$f(x) = \langle \theta, \phi(x) \rangle_{\mathcal{H}} \tag{9}$$

Introduction to Kernel methods

Replace inputs $x \in \mathcal{X}$ by a function $\phi(x) \in \mathcal{H}$, with $\mathcal{H}$ a $\mathbb{R}$-Hilbert space. We then perform linear predictions on $\phi(x)$. This means that estimators have the form :

$$f(x) = \langle \theta, \phi(x) \rangle_{\mathcal{H}} \tag{10}$$

- $\langle ., . \rangle_{\mathcal{H}}$: inner product defined on $\mathcal{H}$. When there is no ambiguity, we will note $\langle ., . \rangle = \langle ., . \rangle_{\mathcal{H}}$.
- $\theta \in \mathcal{H}$
- $\phi(x)$: **feature** associated to x, $\mathcal{H}$: **feature space** .

Interest

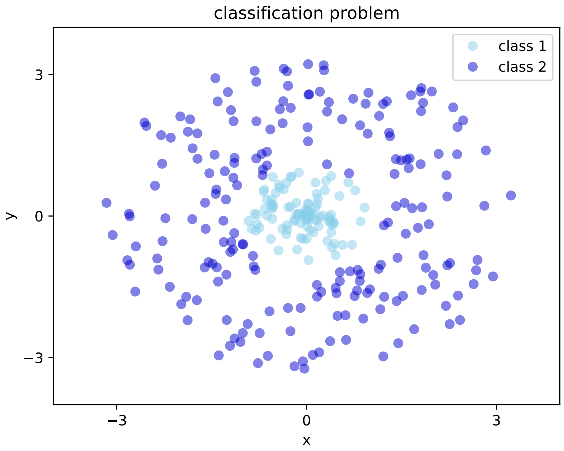
- Kernel methods provide stable algorithms, with theoretical convergence guarantees.
- They can benefit from the smoothness (regularity) of the target function, whereas local averaging methods cannot.
- They can be applied in high dimension.

In some supervised learning problems with many observations, such as computer vision and natural language processing, they are now outperformed by neural networks.

Feature maps

- $\mathcal{X}$ need not be a vector space.
- $\phi(x)$ can provide more useful **representation** of the input for the considered problem (classification, regression).
- The prediction function is then allowed to depend **non-linearly** on x.

Nonlinear data separation



Feature space

Often, $\mathcal{H} = \mathbb{R}^d$, but importantly, we will see that d can even be **infinite**, thanks to a computation trick called the **kernel trick**.

Representer theorem

We consider a framework where we look for a minimizer $\hat{\theta}$ of a loss such as

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(y_i, \langle \theta, \phi(x_i) \rangle) + \lambda ||\theta||^2 \tag{11}$$

The key aspect is that the input observations $x_i \in \mathcal{X}$ are only accessed through inner products with θ.

Representer theorem

Theorem

Let $\Psi : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$ be a strictly increasing function with respect to the last variable. Then, the minimum of

$$L(\theta) = \Psi(\langle \theta, \phi(x_i) \rangle, \dots, \langle \theta, \phi(x_n) \rangle, ||\theta||^2) \tag{12}$$

is attained for $\hat{\theta} \in Vect(\{\phi(x_i)\}$. We can write

$$\hat{\theta} = \sum_{i=1}^{n} \alpha_i \phi(x_i), \alpha \in \mathbb{R}^n \tag{13}$$

Proof I

Let $\mathcal{H}_D = \{\sum_{i=1}^n \alpha_i \phi(x_i), \alpha \in \mathbb{R}^n\}$. For all $\theta \in \mathcal{H}$, we have a decomposition

$$\theta = \theta_D + \theta_{D^\perp} \tag{14}$$

with $\theta_D \in \mathcal{H}_D$ and $\theta_{D^\perp} \in \mathcal{H}_D^\perp$.
Then, $\forall i \in \{1, \dots, n\}$,

$$\begin{aligned} \langle \theta, \phi(x_i) \rangle &= \langle \theta_D, \phi(x_i) \rangle + \langle \theta_{D^\perp}, \phi(x_i) \rangle \\ &= \langle \theta_D, \phi(x_i) \rangle \end{aligned} \tag{15}$$

Furthermore,

$$||\theta||^2 = ||\theta_D||^2 + ||\theta_{D^\perp}||^2 \tag{16}$$

Proof II

Hence

$$\begin{aligned}
&\Psi(\langle\theta,\phi(x_i)\rangle,\dots,\langle\theta,\phi(x_n)\rangle,||\theta||^2)\\
&=\Psi(\langle\theta_D,\phi(x_i)\rangle,\dots,\langle\theta_D,\phi(x_n)\rangle,||\theta_D||^2+||\theta_{D^\perp}||^2)\\
&\geq \Psi(\langle\theta_D,\phi(x_i)\rangle,\dots,\langle\theta_D,\phi(x_n)\rangle,||\theta_D||^2)
\end{aligned} \tag{17}$$

This means that

$$\begin{aligned}
&\inf_{\theta\in\mathcal{H}} \Psi(\langle\theta,\phi(x_i)\rangle,\dots,\langle\theta,\phi(x_n)\rangle,||\theta||^2)\\
&=\inf_{\theta\in\mathcal{H}_D} \Psi(\langle\theta,\phi(x_i)\rangle,\dots,\langle\theta,\phi(x_n)\rangle,||\theta||^2)
\end{aligned} \tag{18}$$

Application to supervised learning

The loss

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, \langle \theta, \phi(x_i) \rangle + \lambda ||\theta||^2 \qquad (19)$$

is of the form Ψ and is increasing in the last variable.
As a direct consequence, the minimum of 19 is attained at

$$\theta = \sum_{i=1}^{n} \alpha_i \phi(x_i), \alpha \in \mathbb{R}^n \qquad (20)$$

We note that no convexity hypothesis on l is required.

Consequence

We note

- α the vector such that $\theta = \sum_{i=1}^{n} \alpha_i \phi(x_i)$.
- $K \in \mathbb{R}^{n,n}$ the matrix defined by

$$K_{ij} = \langle \phi(x_i), \phi(x_j) \rangle$$

Consequence

We remark that $\forall i \in [1, n]$,

$$\begin{aligned}\langle \theta, \phi(x_i) \rangle &= \sum_{j=1}^{n} \alpha_j \langle \phi(x_j), \phi(x_i) \rangle \\ &= \sum_{j=1}^{n} \alpha_j K_{ij} \\ &= (K\alpha)_i\end{aligned}$$

And we also remark that

$$\begin{aligned}
||\theta||^2 &= \langle \theta, \theta \rangle \\
&= \langle \sum_{i=1}^{n} \alpha_i \phi(x_i), \sum_{i=j}^{n} \alpha_j \phi(x_j) \rangle \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle \phi(x_i), \phi(x_j) \rangle \\
&= \sum_{i=1}^{n} \alpha_i (\sum_{j=1}^{n} K_{ij} \alpha_j) \\
&= \sum_{i=1}^{n} \alpha_i (K\alpha)_i \\
&= \alpha^T K \alpha
\end{aligned}$$

Finally

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(y_i, (K\alpha)_i) + \lambda\alpha^T K\alpha$$

$L(\theta)$ can be written **only** with K and α, instead of $\phi(x_i)$.
Natural question : But this does not make sense, as $\phi(x_i)$ and $\phi(x_j)$ are required to compute $K_{ij} = k(x_i, x_j)$?
Yes, **but**, in some situations, it is possible to compute $k(x_i, x_j)$ **without explicit knowledge** of ϕ. This is known as the **kernel trick**.

Alternate minimization problem

$$\begin{aligned}&\inf_{\theta\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} l(y_i, \langle\theta, \phi(x_i)\rangle) + \lambda||\theta||^2 \\ &= \inf_{\alpha\in\mathbb{R}^n} \frac{1}{n}\sum_{i=1}^{n} l(y_i, (K\alpha)_i) + \lambda\alpha^T K\alpha\end{aligned} \tag{21}$$

Alternate minimization problem

$$\begin{aligned} & \inf_{\theta \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} l(y_i, \langle \theta, \phi(x_i) \rangle) + \lambda ||\theta||^2 \\ = & \inf_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} l(y_i, (K\alpha)_i) + \lambda \alpha^T K \alpha \end{aligned} \tag{22}$$

It might be easier to optimize in $\mathbb{R}^n$ than in $\mathcal{H}$, especially if $\mathcal{H}$ is infinite dimensional.

Evaluation fonction

Also, we can rewrite the evaluation function as :

$$f(x) = \theta^T \phi(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x)$$

Gram matrix

The kernel matrix is a matrix of inner products. It is often called a Gram matrix. If we note the design matrix

$$\phi = \begin{pmatrix} \phi(x_1)^T \\ \dots \\ \phi(x_i)^T \\ \dots \\ \phi(x_n)^T \end{pmatrix}$$

Then

$$K = \phi\phi^T \in \mathbb{R}^{n,n} \tag{23}$$

Gram matrix

K is symmetric positive semi-definite. $\forall \alpha \in \mathbb{R}^n$,

$$\begin{aligned} \alpha K \alpha &= \alpha \phi \phi^T \alpha \\ &= (\phi^T \alpha)^T (\phi^T \alpha) \\ &= ||\phi^T \alpha||^2 \end{aligned} \tag{24}$$

Then, if λ is an eigenvalue of K, with eigenvector α_λ,

$$\begin{aligned} \alpha_\lambda K \alpha_\lambda &= \alpha_\lambda \lambda \alpha_\lambda \\ &= \lambda ||\alpha_\lambda||^2 \end{aligned} \tag{25}$$

Approximations

- when n is large, it can become too costly to compute and store K ($\mathcal{O}(n^2)$) and to solve the optimization problem ($\mathcal{O}(n^3)$).
- to avoid explicitely computing and storing K, **low-rank approximations** may be used (such as Nyström method)
- to solve the optimization problem, **low-rank decomposition** may be used.

See also

- Kernels on structured objects (graphs, texts, etc)
- Reproducing kernel Hilbert space (RKHS) (the space $\mathcal{H}$ that corresponds to k and ϕ)
- Adaptivity of kernel methods to the smoothness of the target function. If the optimization if performed in the right way, the convergence is faster for functions that are more than simply Lipshitz-continuous.

Applications of kernels

Classical frameworks for using kernels :

- Support vector machines (SVMs)
- Kernel ridge regression
- Kernel pca (principal component analysis)

Gaussian kernel (RBF)

$$k(x, x') = \exp\Big(- \gamma ||x - x'||^2 \Big) \tag{26}$$

With $k(x, x') = \langle \phi(x), \phi(x') \rangle$, $\phi(x) \in \mathcal{H}$

- $\mathcal{H}$ is of infinite dimension.
- γ is a parameter that should be carefully tuned.

Linear kernel

$$k(x, x') = \langle x, x' \rangle \tag{27}$$

References I

Bengio, Y. (2009).
CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING.
Foundations and Trends® in Machine Learning.

Mnih, V., Silver, D., and Riedmiller, M. (2013).
Deep Q Network (Google).
Arxiv.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016).

References II

Mastering the Game of Go with Deep Neural Networks and Tree Search.
Nature, 529(7587) :484–489.