CSCI 567: Machine Learning

Vatsal Sharan
Spring 2024

Lecture 12, Apr 12

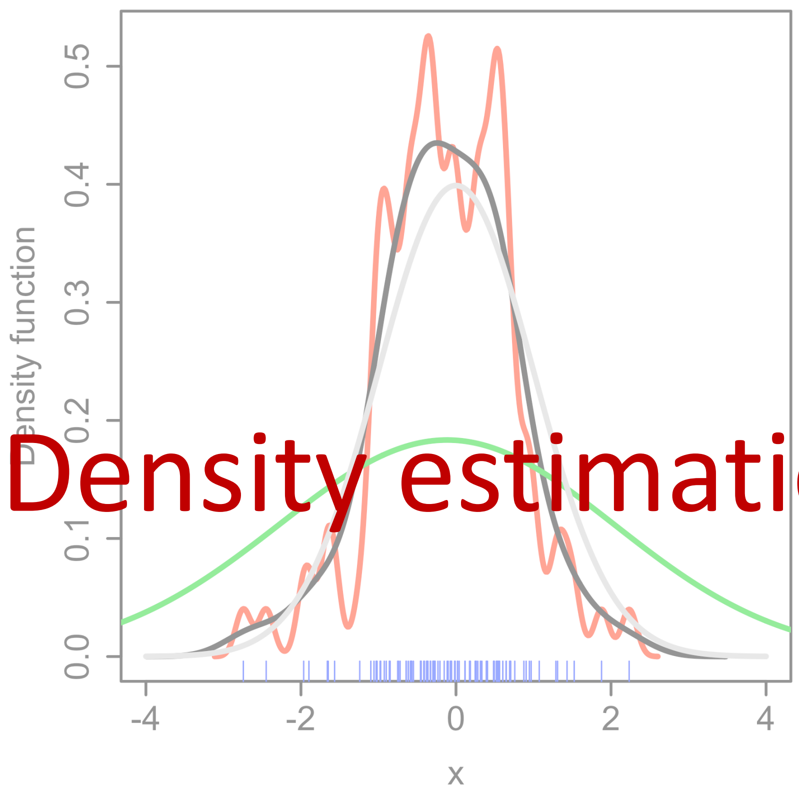
USC University of Southern California

Administrivia

- Exam 2 is on April 26 in class (1pm-3:20pm)
  - Similar format to Exam 1
  - Syllabus is lecture 6 (multiclass classification & neural networks) onwards

- Project mid-term check-ins next week
  - Short report due on Monday April 15 on Gradescope

- Today’s plan:
  - Density estimation & Naïve Bayes
  - Multi-armed bandits
  - Reinforcement Learning



Density estimation

Density estimation

- Introduction
- Parametric methods
- Non-parametric methods

Introduction

With clustering using GMMs, our high-level goal was the following:

Given a training set $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, **estimate a density function p that could have generated this dataset** (via $\boldsymbol{x}_i \overset{i.i.d.}{\sim} p$).

This is a special case of the general problem of *density estimation*, an important unsupervised learning problem.

Density estimation is useful for many downstream applications

- we have seen clustering already, will see more today
- these applications also *provide a way to measure quality of the density estimator*

.

Density estimation

- Introduction
- Parametric methods
- Non-parametric methods

Parametric methods: generative models

Parametric estimation assumes a generative model parametrized by $\boldsymbol{\theta}$:

$$p(\boldsymbol{x}) = p(\boldsymbol{x}\ ; \boldsymbol{\theta})$$

Examples:

- GMM: $p(\boldsymbol{x}\ ; \boldsymbol{\theta}) = \sum_{j=1}^{k} \pi_j N(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ where $\boldsymbol{\theta} = \{\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}$
- Multinomial: a discrete variable with values in $\{1, 2, \ldots, k\}$ s.t.

  $$p(x = j\ ; \boldsymbol{\theta}) = \theta_j$$

  where $\boldsymbol{\theta}$ is a distribution over the k elements.

Size of $\boldsymbol{\theta}$ is independent of the size of the training set, so it's parametric.

Parametric methods: estimation

As usual, we can apply **MLE** to learn the parameters $\boldsymbol{\theta}$:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \underbrace{\sum_{i=1}^{n} \ln p(x_i\ ; \boldsymbol{\theta})}_{\hookrightarrow \text{ iid assumption}}$$

For some cases this is intractable and we can use algorithms such as EM to approximately solve the MLE problem (e.g. GMMs).

For some other cases this admits a simple closed-form solution (e.g. multinomial).

MLE for multinomials

The log-likelihood is

$$\sum_{i=1}^{n} \ln p(x = x_i\, ; \boldsymbol{\theta}) = \sum_{i=1}^{n} \ln \theta_{x_i} = \sum_{i=1}^{n} \sum_{j=1}^{k} \underbrace{\mathbf{1}(x_i = j)}_{\text{indicator}} \ln \theta_j$$

$$= \sum_{j=1}^{k} \sum_{i:x_i=j} \ln \theta_j = \sum_{j=1}^{k} z_j \ln \theta_j$$

where $z_j = |\{i : x_i = j\}|$ is **the number of examples with value j**.

The solution is simply

$$\theta_j = \frac{z_j}{n} \propto z_j,$$

i.e. the fraction of examples with value j. (See HW4 Q2.1)

Density estimation

- Introduction
- Parametric methods
- Non-parametric methods

Nonparametric methods

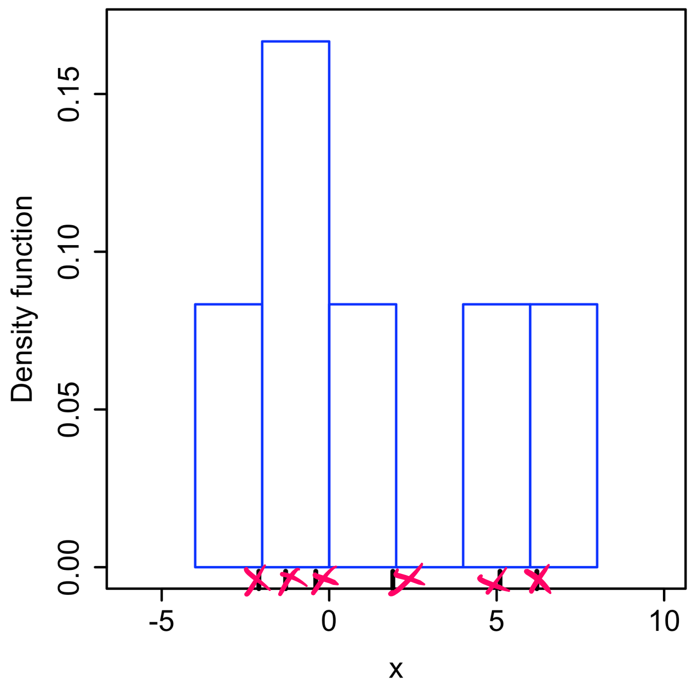
Can we estimate *without assuming a fixed generative model?*

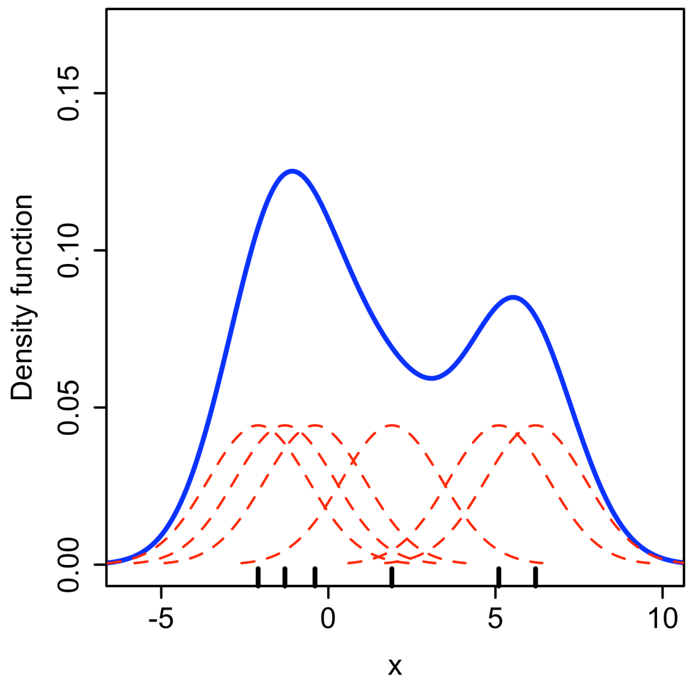
Kernel density estimation (KDE) provides a solution.

- the approach is nonparametric: it keeps the entire training set
- we focus on the one-dimensional (continuous) case

High-level idea

- Construct something similar to a histogram:
- For each data point, create a “bump” (via a Kernel)
- Sum up or average all the bumps





Kernel

KDE with **a kernel** $K: \mathbb{R} \to \mathbb{R}$:

$$\int_{-\infty}^{\infty} p(x)\,dx = 1 \leftarrow \quad p(x) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{K(x - x_i)}_{\text{adding a bump around } x_i}$$

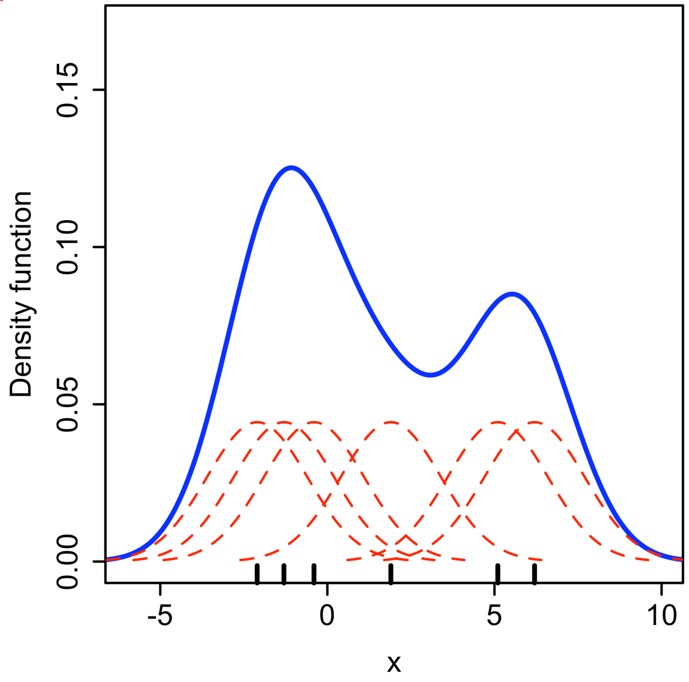
→ need to keep around all datapoints

e.g. $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}$, the standard Gaussian density

↘ bump

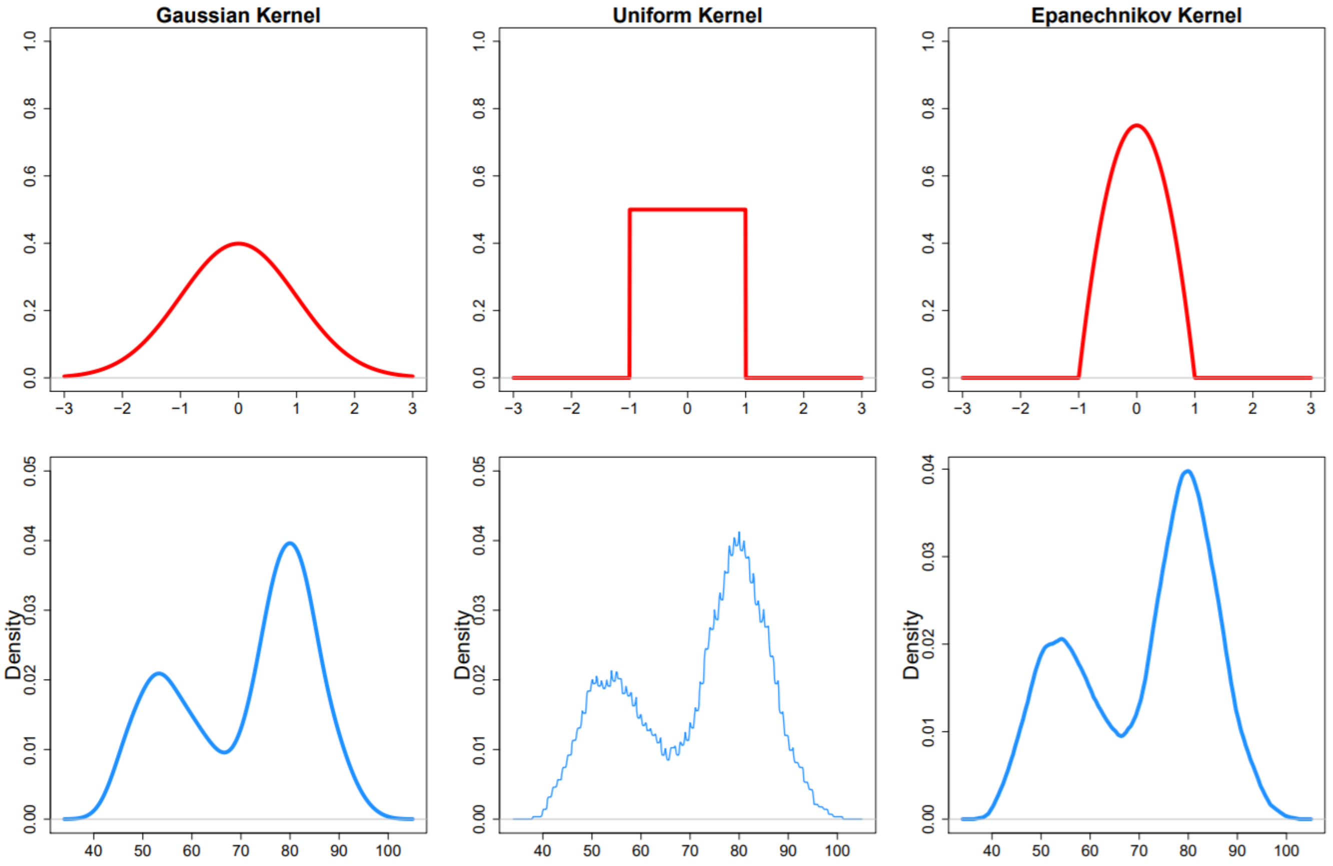
Kernel needs to satisfy:

- symmetry: $K(u) = K(-u)$
- $\int_{-\infty}^{\infty} K(u)du = 1$, makes sure p is a density function.

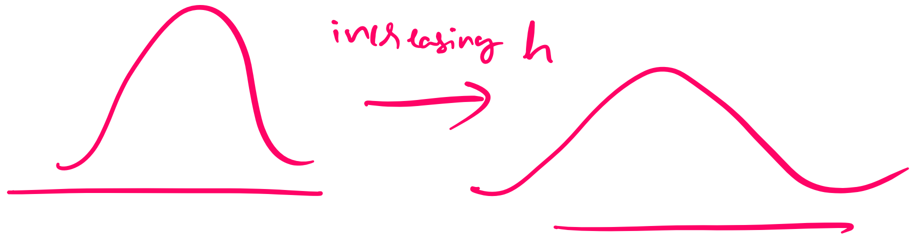


Different kernels

$$\frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} \qquad \frac{1}{2}\mathbb{I}[|u| \leq 1] \qquad \frac{3}{4}\max\{1-u^2, 0\}$$



Bandwidth



If $K(u)$ is a kernel, then for any $h > 0$

$$K_h(u) := \frac{1}{h} K\left(\frac{u}{h}\right)$$

(stretching the kernel)

can be used as a kernel too (verify the two properties yourself)

$$\int_{-\infty}^{\infty} \frac{1}{h} K\left(\frac{u}{h}\right) du = 1$$

So general KDE is determined by both the kernel K and the bandwidth h

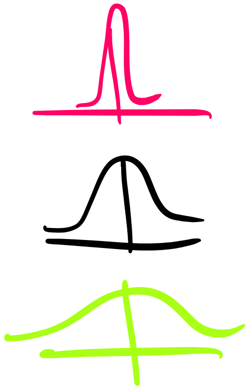
$$p(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

- x_i controls the center of each bump
- h controls the width/variance of the bumps

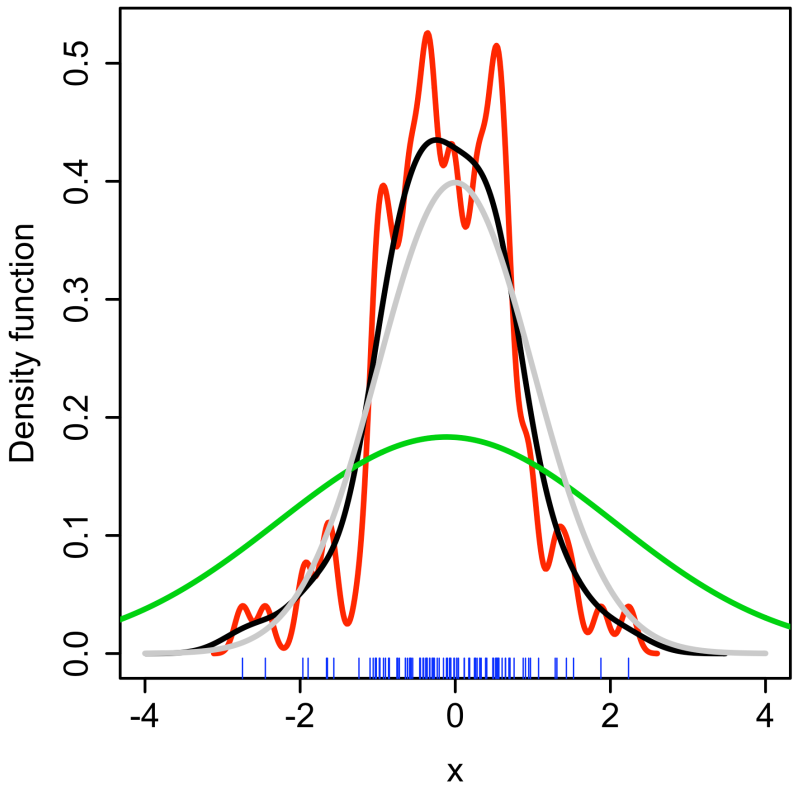
Bandwidth

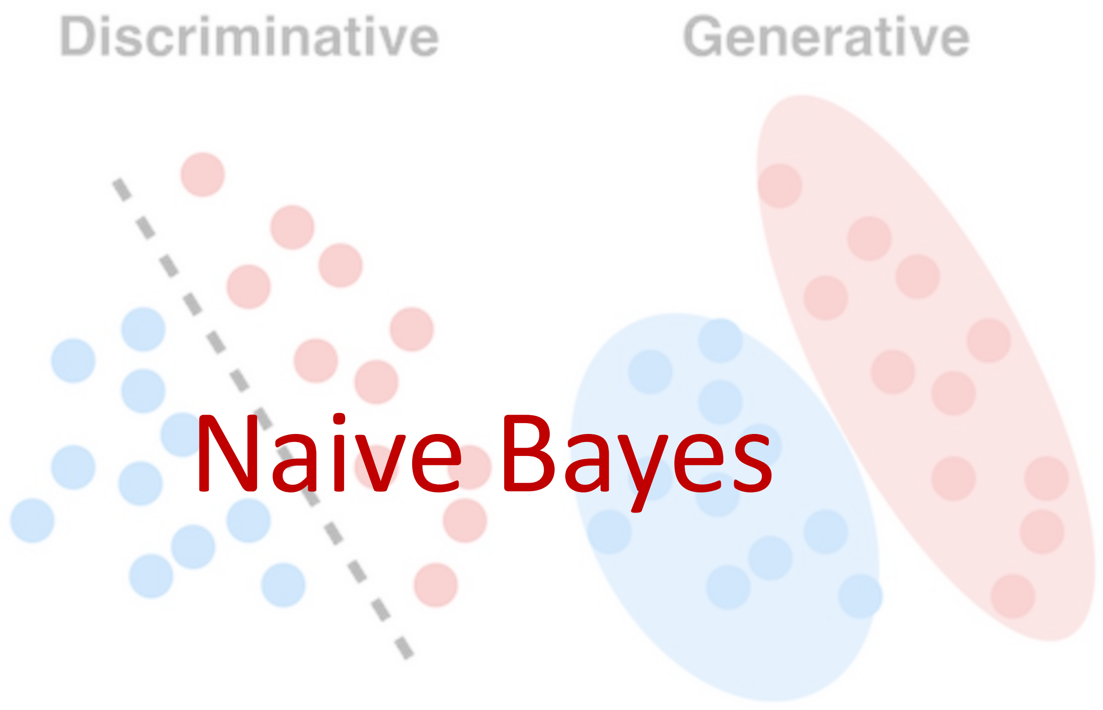
Larger h means larger variance and smoother density

Gray curve is ground-truth



- Red: $h = 0.05$
- Black: $h = 0.337$
- Green: $h = 2$





Naive Bayes

Image source

A simplistic taxonomy of ML

Supervised learning:
Aim to predict outputs of future datapoints

Unsupervised learning:
Aim to discover hidden patterns and explore data

Reinforcement learning:
Aim to make sequential decisions

Naïve Bayes

- Motivation & setup
- Prediction with Naïve Bayes, and some connections

Bayes optimal classifier

Suppose $(\boldsymbol{x}, y)$ is drawn from a joint distribution p. The **Bayes optimal classifier** is

$$f^*(\boldsymbol{x}) = \operatorname*{argmax}_{c \in [\mathsf{C}]} p(c \mid \boldsymbol{x})$$

i.e. predict the class with the largest conditional probability.

p is of course unknown, but we can estimate it, which is *exactly a density estimation problem!*

Estimation

How to estimate a joint distribution? Observe we always have

$$p(\boldsymbol{x}, y) = p(y)p(\boldsymbol{x} \mid y)$$

We know how to estimate $p(y)$ by now. $\rightarrow$ C possible values

To estimate $p(\boldsymbol{x} \mid y = c)$ for some $c \in [\mathsf{C}]$, we are doing density estimation using data $\{\boldsymbol{x}_i : y_i = c\}$.

This is *not a 1D problem* in general.

A naïve assumption

Naive Bayes assumption: **conditioning on a label, features are independent**, which means

$$p(\boldsymbol{x} \mid y = c) = \prod_{j=1}^{d} p(x_j \mid y = c)$$

Now for each j and c we have a simple 1D density estimation problem!

Is this a reasonable assumption? Sometimes yes, e.g.

- use $\boldsymbol{x} =$ (Height, Vocabulary) to predict $y =$ Age
- Height and Vocabulary are dependent
- but conditioned on Age, they are independent!

More often this assumption is *unrealistic and "naive"*, but still Naive Bayes can work very well even if the assumption is wrong.

Example: Discrete features

Height: ≤3', 3'-4', 4'-5', 5'-6', ≥6' $\nearrow$ 5 possible values

Vocabulary: ≤5K, 5K-10K, 10K-15K, 15K-20K, ≥20K

Age: ≤5, 5-10, 10-15, 15-20, 20-25, ≥25 $\rightarrow$ 6 values

MLE estimation: e.g.

$$p(\text{Age} = \text{10-15}) = \frac{\text{\#examples with age 10-15}}{\text{\#examples}}$$

$$p(\text{Height} = \text{5'-6'} \mid \text{Age} = \text{10-15})$$
$$= \frac{\text{\#examples with height 5'-6' and age 10-15}}{\text{\#examples with age 10-15}}$$

$\rightarrow$ conditional $= \frac{\text{joint}}{\text{marginal}}$

Discrete features: More formally

For a label $c \in [C]$,

$$p(y = c) = \frac{|\{i : y_i = c\}|}{n}$$

For each possible value ℓ of a discrete feature j,

jth feature of ith datapoint

$$\underbrace{p(x_j = \ell \mid y = c)}_{\text{density for jth feature}} = \frac{|\{i : \overset{\swarrow}{x_{i,j}} = \ell, y_i = c\}|}{|\{i : y_i = c\}|}$$

Continuous features

If the feature is continuous, we can do

- parametric estimation, e.g. via a Gaussian

$$p(x_j = x \mid y = c) = \frac{1}{\sqrt{2\pi}\sigma_{c,j}} \exp\left(-\frac{(x-\mu_{c,j})^2}{2\sigma_{c,j}^2}\right)$$

  where $\mu_{c,j}$ and $\sigma_{c,j}^2$ are the empirical mean and variance of feature j among all examples with label c.

- or nonparametric estimation, e.g. via a Kernel K and bandwidth h:

$$p(x_j = x \mid y = c) = \frac{1}{|\{i : y_i = c\}|} \sum_{i:y_i=c} K_h(x - x_{i,j})$$

Naïve Bayes

- Motivation & setup
- Prediction with Naïve Bayes, and some connections

Naïve Bayes: Prediction

After learning the model

$$p(\boldsymbol{x}, y) = p(y) \prod_{j=1}^{d} p(x_j \mid y)$$

the **prediction** for a new example $\boldsymbol{x}$ is

$$\operatorname*{argmax}_{c \in [\mathsf{C}]} \; p(y = c \mid \boldsymbol{x}) = \operatorname*{argmax}_{c \in [\mathsf{C}]} \; p(\boldsymbol{x}, y = c) \quad = p(x) \cdot p(y = c \mid x)$$

$$= p(y=c) \cdot p(x \mid y = c)$$

$$= \operatorname*{argmax}_{c \in [\mathsf{C}]} \left(\underbrace{p(y = c)}_{\text{prior}} \underbrace{\prod_{j=1}^{d} p(x_j \mid y = c)}_{\text{likelihood}} \right) \rightarrow \text{Naive Bayes}$$

$$= \operatorname*{argmax}_{c \in [\mathsf{C}]} \left(\ln p(y = c) + \sum_{j=1}^{d} \ln p(x_j \mid y = c) \right).$$

Example: Discrete features

For **discrete features**, plugging in previous MLE estimation gives

$$\begin{aligned} &\operatorname*{argmax}_{c\in[\mathsf{C}]} \; p(y=c \mid \boldsymbol{x}) \\ &= \operatorname*{argmax}_{c\in[\mathsf{C}]} \left(\ln p(y=c) + \sum_{j=1}^{d} \ln p(x_j \mid y=c) \right) \\ &= \operatorname*{argmax}_{c\in[\mathsf{C}]} \left(\ln |\{i : y_i = c\}| + \sum_{j=1}^{d} \ln \frac{|\{i : x_{i,j} = x_j, y_i = c\}|}{|\{i : y_i = c\}|} \right) \end{aligned}$$

What is Naïve Bayes learning?

Observe again for the case of continuous features with a Gaussian model, if we **fix the variance for each feature to be σ** (i.e. not a parameter of the model any more), then the prediction becomes

$$\operatorname*{argmax}_{c\in[\mathsf{C}]} p(y=c \mid \boldsymbol{x})$$

$$= \operatorname*{argmax}_{c\in[\mathsf{C}]} \left(\ln|\{i : y_i = c\}| + \underbrace{\sum_{j=1}^{d} \left(\ln\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{(x_j - \mu_{c,j})^2}{2\sigma^2} \right)}_{\text{Gaussian density}} \right)$$

x_j^2 is constant across classes

$$= \operatorname*{argmax}_{c\in[\mathsf{C}]} \left(\underbrace{\ln|\{i : y_i = c\}| - \sum_{j=1}^{d} \frac{\mu_{c,j}^2}{2\sigma^2}} + \sum_{j=1}^{d} \frac{\mu_{c,j}}{\sigma^2} x_j \right)$$

$$= \operatorname*{argmax}_{c\in[\mathsf{C}]} \left(w_{c0} + \sum_{j=1}^{d} w_{cj} x_j \right) = \operatorname*{argmax}_{c\in[\mathsf{C}]} \boldsymbol{w}_c^{\mathrm{T}} \boldsymbol{x} \longrightarrow \text{multiclass linear classification}$$

$w_c = [w_{c0}, w_{c1}, \ldots, w_{cd}]$

where we denote $w_{c0} = \ln|\{i : y_i = c\}| - \sum_{j=1}^{d} \frac{\mu_{c,j}^2}{2\sigma^2}$ and $w_{cj} = \frac{\mu_{c,j}}{\sigma^2}$.

Connection to logistic regression

Moreover, by a similar calculation you can verify

$$p(y = c \mid \boldsymbol{x}) \propto e^{\boldsymbol{w}_c^{\mathrm{T}} \boldsymbol{x}}$$

This the **softmax** function, *the same model we used for the probabilistic interpretation of logistic regression!*

So what is different then? They **learn the parameters in different ways**:

- both via MLE, one on $p(y = c \mid \boldsymbol{x})$, the other on $p(\boldsymbol{x}, y)$
- solutions are different: logistic regression has no closed-form, naive Bayes admits a simple closed-form

Connection to logistic regression

	Logistic regression	Naive Bayes
Model	conditional $p(y \mid \boldsymbol{x})$	joint $p(\boldsymbol{x}, y)$
Learning	MLE (can also be viewed as minimizing logistic loss)	MLE
Accuracy	usually better for large n	usually better for small n

$p(y=c \mid x)$ for $c = \{\pm 1\}$

$= \sigma(y w^T x)$

Discriminative models

Generative models

Multiarmed bandits

A simplistic taxonomy of ML

Supervised learning:
Aim to predict outputs of future datapoints

Unsupervised learning:
Aim to discover hidden patterns and explore data

Reinforcement learning:
Aim to make sequential decisions

Multi-armed bandits

- Motivation & setup
- Exploration vs. Exploitation

Decision making

Problems we have discussed so far:

- start with a fixed training dataset
- learn a predictor from the data or discover some patterns in the data

But many real-life problems are about **learning continuously**:

- make a prediction/decision
- receive some feedback
- repeat

Broadly, these are called **online decision making problems**.

Examples

Amazon/Netflix/Instagram **recommendation systems**:

- a user visits the website (or views a post etc.)
- the system recommends some products/movies/posts
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

- make a move
- receive some reward (e.g. score a point) or loss (e.g. fall down)
- make another move...

Multiarmed bandits: Motivation

Imagine going to a casino to play a slot machine

- it robs you, like a “bandit” with a single arm

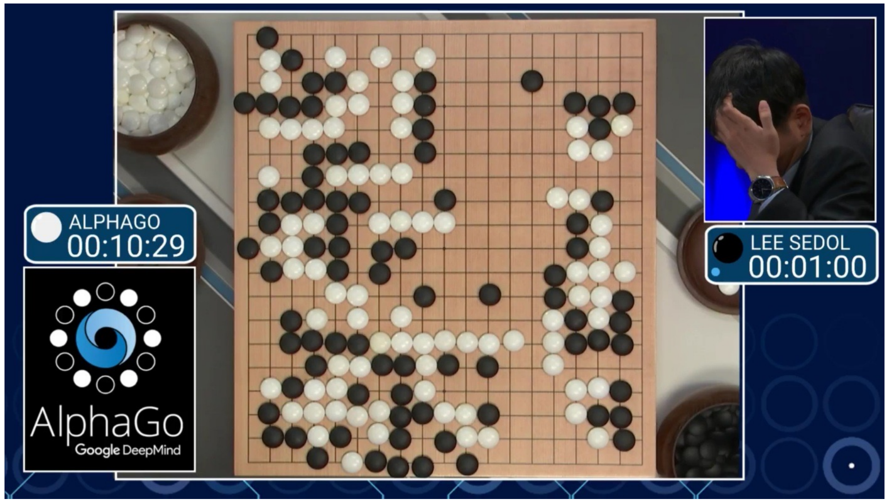
Of course there are many slot machines in the casino

- like a bandit with multiple arms (hence the name)
- **if I can play 10 times, which machines should I play?**

Applications

This simple model and its variants capture many real-life applications:

- recommendation systems, each product/movie/news story is an arm (Netflix employs a variant of bandit algorithm)
- game playing, each possible move is an arm (AlphaGo has a bandit algorithm as one of the components)



Formal setup

There are K **arms** (actions/choices/...)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$
- the learner observes the reward for arm a_t, i.e., r_{t,a_t}

$\uparrow$ reward for the k-th arm at time t.

Importantly, *learner does not observe rewards for arms not selected!*

This kind of limited feedback is usually referred to as bandit feedback

Evaluating performance

What should be the goal here?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural.

But the absolute value of rewards is not meaningful, instead we should compare it to some *benchmark*. A classic benchmark is

$$\max_{a\in[K]} \sum_{t=1}^{T} r_{t,a}$$

i.e. the largest reward one can achieve by always playing a fixed arm

So we want to minimize

$$\max_{a\in[K]} \sum_{t=1}^{T} r_{t,a} - \sum_{t=1}^{T} r_{t,a_t}$$

This is called the **regret**: *how much I regret not sticking with the best fixed arm in hindsight?*

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution
- they could be generated via some changing distribution
- they could be generated even completely arbitrarily/adversarially

We focus on a simple setting:

- rewards $\overset{\{0,1\}}{\text{of}}$ arm a are i.i.d. samples of Ber(μ_a), that is, $r_{t,a}$ is 1 with prob. μ_a, and 0 with prob. $1 - \mu_a$, independent of anything else. (indep. across all arms, across timesteps)
- each arm has a different mean $(\mu_1, \ldots, \mu_K)$; the problem is essentially about finding the best arm $\operatorname{argmax}_a \mu_a$ as quickly as possible

Empirical means

Let $\hat{\mu}_{t,a}$ be the **empirical mean** of arm a up to time t:

$$\hat{\mu}_{t,a} = \frac{1}{n_{t,a}} \sum_{\tau \leq t : a_\tau = a} r_{\tau,a}$$

where

$$n_{t,a} = \sum_{\tau \leq t} \mathbb{I}[a_\tau == a]$$

is the **number of times** we have picked arm a.

Concentration: $\hat{\mu}_{t,a}$ should be close to μ_a if $n_{t,a}$ is large

Multi-armed bandits

- Motivation & setup
- Exploration vs. Exploitation

Exploitation only

Greedy:

Pick each arm once for the first K rounds.

For $t = K+1, \ldots, T$, pick $a_t = \mathrm{argmax}_a \; \hat{\mu}_{t-1,a}$.

What's wrong with this greedy algorithm?

Consider the following example:

- $K = 2, \mu_1 = 0.6, \mu_2 = 0.5$ (so arm 1 is the best)
- suppose the algorithm first picks arm 1 and sees reward 0, then picks arm 2 and sees reward 1 (this happens with decent probability) $(0.4 \times 0.5 = 0.2)$
- the algorithm will never pick arm 1 again!

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

- on one hand we want to exploit the arms that we think are good
- on the other hand we need to explore all arms often enough in order to figure out which one is better
- so each time we need to ask: *do I explore or exploit? and how?*

We next discuss **three ways** to trade off exploration and exploitation for our simple multi-armed bandit setting.

A natural first attempt

Explore–then–Exploit:

Input: a parameter $T_0 \in [T]$

Exploration phase: for the first T_0 rounds, pick each arm for T_0/K times

Exploitation phase: for the remaining $T - T_0$ rounds, stick with the empirically best arm $\text{argmax}_a \ \hat{\mu}_{T_0,a}$

Parameter T_0 clearly controls the exploration/exploitation trade-off

Explore-then-Exploit: Issues

It's pretty reasonable, but the disadvantages are also clear:

- not clear how to tune the hyperparameter T_0
- in the exploration phase, even if an arm is clearly worse than others based on a few pulls, it's still pulled T_0/K times
- clearly it won't work if the environment is changing

A slightly better algorithm

ϵ-Greedy Pick each arm once for the first K rounds.

For $t = K+1, \dots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1-\epsilon$, exploit: pick $a_t = \operatorname{argmax}_a \ \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

Cons

- need to tune ϵ
- same uniform exploration

Is there a more adaptive way to explore?

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \text{UCB}_{t,a}$ where

$$\text{UCB}_{t,a} := \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

initialize all estimates to same value

if $n_{t-1,a}$ is small, $\text{UCB}_{t,a}$ will be large

- the first term in $\text{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration

- the bonus term is large if the arm is not pulled often enough, which encourages exploration (adaptive due to the first term)

- a parameter-free algorithm, and *it enjoys optimal regret!*

Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

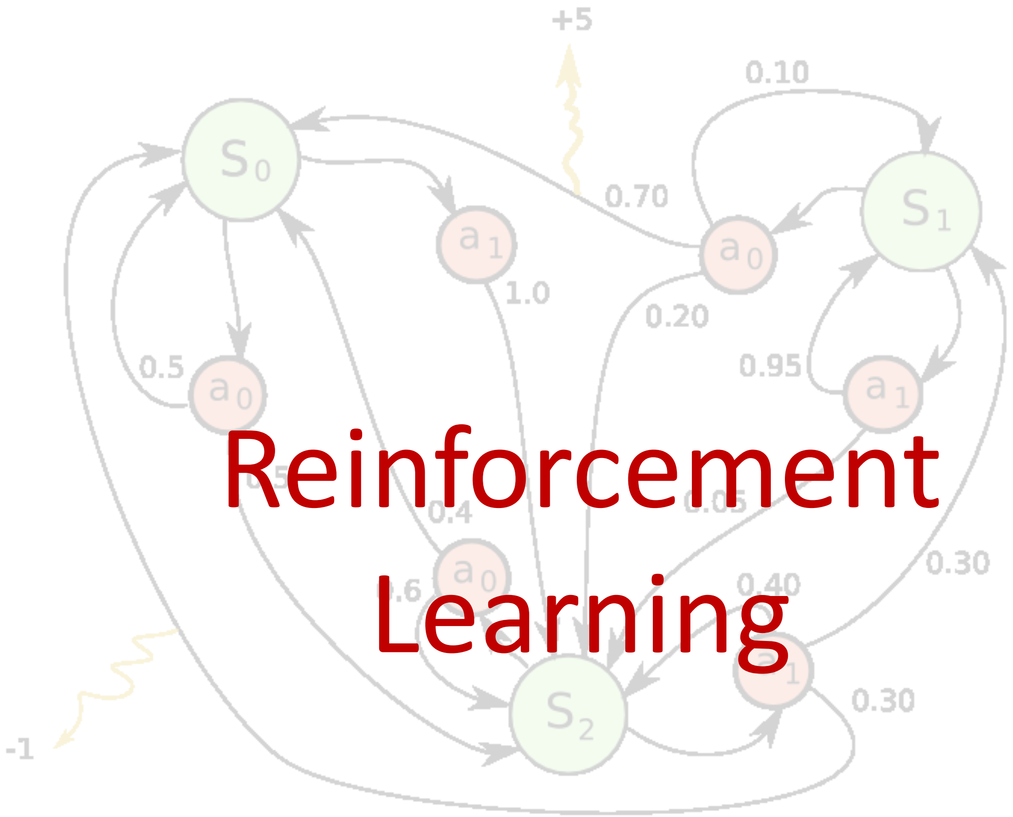
$$\mu_a \leq \mathrm{UCB}_{t,a}$$

so $\mathrm{UCB}_{t,a}$ is indeed an upper bound on the true mean.

Another way to interpret UCB, "**optimism in face of uncertainty**":

- true environment (best mean) is unknown due to randomness (**uncertainty**)
- have an upper bound (optimistic guess) on the expected reward of each environment, and pick best one according to upper bound (**optimism**)

This principle is useful for many other bandit problems.



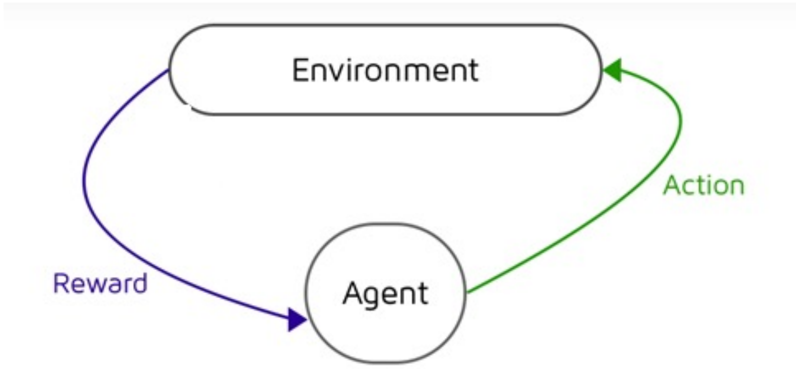
Reinforcement Learning

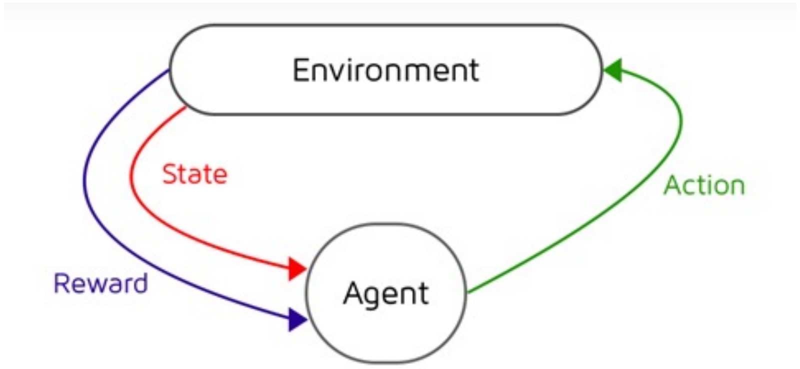
Reinforcement Learning

- Motivation
- Markov Decision Process (MDP)
- Learning MDPs

Motivation

Multi-armed bandit is among the simplest decision making problems with limited feedback.





Often, it can be too simple to capture real-life problems. One important aspect it fails to capture is the "state" of the learning agent, which has impacts on the reward of each action.

- e.g. for Atari games, after making one move, the agent moves to a different state, with possible different rewards for each action

Reinforcement Learning

Reinforcement learning (RL) is one way to deal with this issue.

Huge recent success when combined with deep learning techniques

- Video games, AlphaGo, Reinforcement Learning from Human Feedback (RLHF) for Chat-bots, self-driving cars, etc.

The foundation of RL is **Markov Decision Process (MDP)**,
a combination of Markov models and multi-armed bandits.

Reinforcement Learning

- Motivation
- Markov Decision Process (MDP)
- Learning MDPs

Markov Decision Process

An MDP is parameterized by five elements

- $\mathcal{S}$: a set of possible states
- $\mathcal{A}$: a set of possible actions
- P: transition probability, $P_a(s, s')$ is the probability of transiting from state s to state s' after taking action a (Markov property) $\rightarrow$ Conditioned on present state, future states do not depend on past states
- r: reward function, $r_a(s)$ is (expected) reward of action a at state s

Difference from Markov models, the state transition is influenced by the taken action.

Difference from Multi-armed bandit, the reward depends on the state.

Measuring reward

There can be different ways of measuring reward, for example:

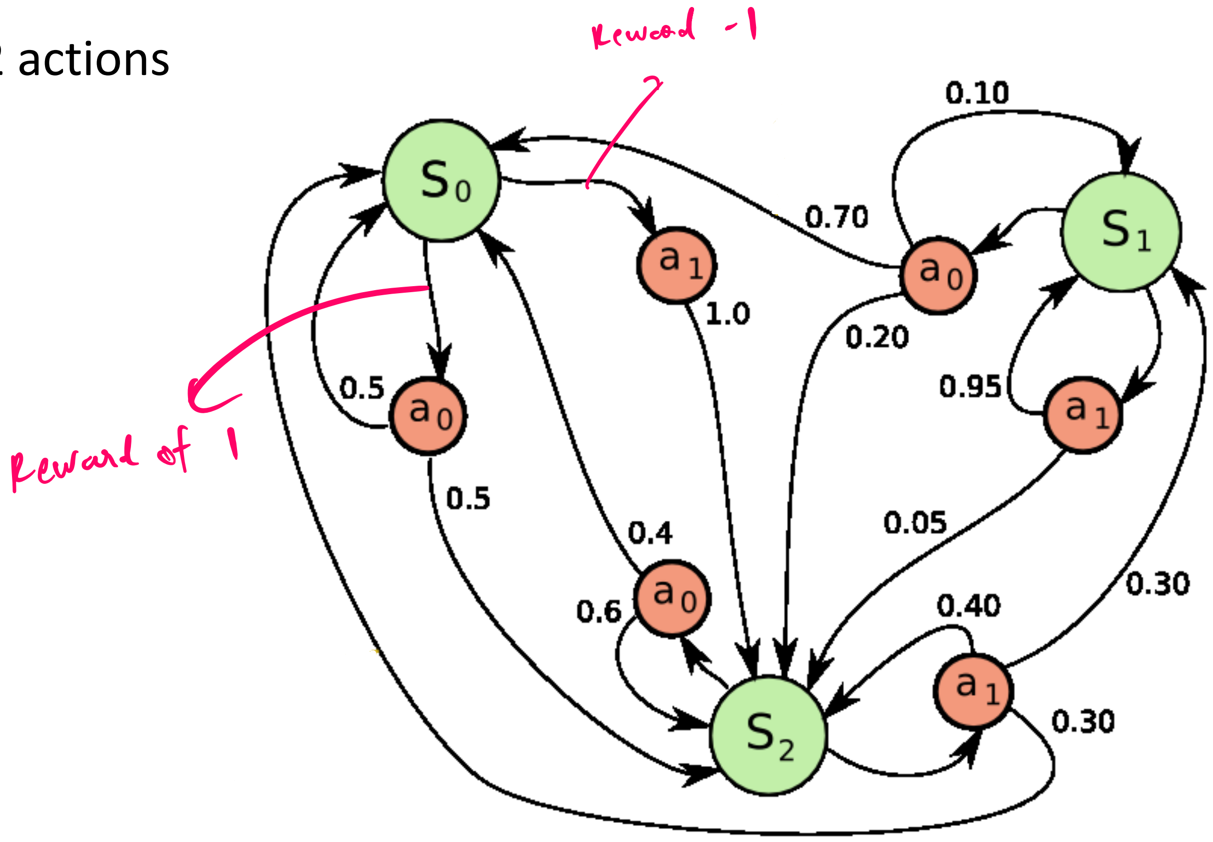
- Fix some finite horizon (say 100 time steps) and sum total the rewards obtained in this horizon.
- Discount future rewards by discount factor $\gamma \in (0, 1)$. Reward of 1 from tomorrow is only counted as γ for today, more generally reward n time steps away is discounted by γ^n.

Discounting is most popular and well-studied.

- It models the fact that the process could stop at any time step
- It has a preference for solutions which are rewarding over short time scales
- It is like a smoothed version of a fixed horizon.

Example

3 states, 2 actions



Policy

A **policy** $\pi : \mathcal{S} \to \mathcal{A}$ indicates which action to take at each state.

If we start from state $s_0 \in \mathcal{S}$ and act according to a policy π, the discounted rewards for time $0, 1, 2, \ldots$ are respectively

$$r_{\pi(s_0)}(s_0), \quad \gamma r_{\pi(s_1)}(s_1), \quad \gamma^2 r_{\pi(s_2)}(s_2), \quad \cdots$$

where $s_1 \sim P_{\pi(s_0)}(s_0, \cdot), \quad s_2 \sim P_{\pi(s_1)}(s_1, \cdot), \quad \cdots$

If we follow the policy forever, the total (discounted) reward is

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t)\right]$$

where the randomness is from $s_{t+1} \sim P_{\pi(s_t)}(s_t, \cdot)$.

Note: the discount factor allows us to consider an infinite learning process

Optimal policy and Bellman equation

First goal: knowing all parameters, *how do we find the optimal policy*

$$\operatorname*{argmax}_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t)\right] \quad ?$$

We first answer a related question: *what is the maximum reward one can achieve starting from an arbitrary state s?*

$$\begin{aligned} V(s) &= \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t) \;\middle|\; s_0 = s\right] \\ &= \max_{a \in \mathcal{A}} \left(r_a(s) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V(s') \right) \end{aligned}$$

V is called the **(optimal) value function**.

It satisfies the above **Bellman equation**: $|\mathcal{S}|$ nonlinear equations with $|\mathcal{S}|$ unknowns, *how do we solve it?*

Value iteration

Value Iteration:

Initialize $V_0(s)$ randomly for all $s \in \mathcal{S}$

For $k = 1, 2, \ldots$ (until convergence)

previous estimate

$$V_k(s) = \max_{a \in \mathcal{A}} \left(r_a(s) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') \overset{\nearrow}{V_{k-1}}(s') \right) \qquad \text{(\textbf{Bellman upate})}$$

Knowing V, the optimal policy π^* is simply

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left(r_a(s) + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V(s') \right)$$

Convergence of value iteration

Does Value Iteration always find the true value function V ? **Yes!**

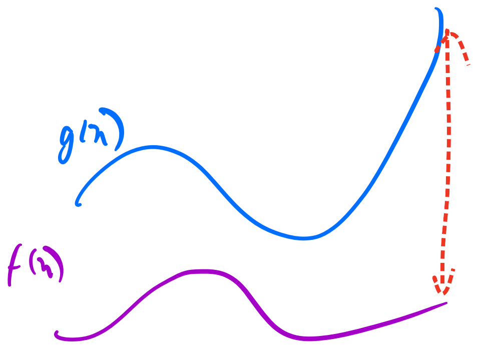
$$|V_k(s) - V(s)| = \left| \max_{a\in\mathcal{A}} \left(r_a(s) + \gamma \sum_{s'\in\mathcal{S}} P_a(s,s')V_{k-1}(s') \right) - \max_{a\in\mathcal{A}} \left(r_a(s) + \gamma \sum_{s'\in\mathcal{S}} P_a(s,s')V(s') \right) \right|$$

$$\leq \gamma \max_{a\in\mathcal{A}} \left| \sum_{s'\in\mathcal{S}} P_a(s,s') \left(V_{k-1}(s') - V(s')\right) \right|$$

$$\leq \gamma \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P_a(s,s') \left|V_{k-1}(s') - V(s')\right|$$

$$\leq \gamma \max_{s'} |V_{k-1}(s') - V(s')| \leq \cdots \leq \gamma^k \max_{s'} |V_0(s') - V(s')|$$

So the distance between V_k and V is shrinking *exponentially fast*.



$$\left|\max_x g(x) - \max_x f(x)\right| \leq \max_x |g(x) - f(x)|$$

$$\left|\sum_i \beta_i\right| \leq \sum_i |\beta_i| \quad \text{for any } \beta_i$$

Reinforcement Learning

- Motivation
- Markov Decision Process (MDP)
- Learning MDPs $\longrightarrow$ Next time!